A Programming Environment for Ubiquitous Computing Environment

Minkyoung Oh[1], Jiyeon Lee[1], Byeong-Mo Chang[1], Joonseon Ahn[2], and Kyung-Goo Doh[3]

[1] Sookmyung Women's University, Seoul, 140-742, Korea, {omk,jiyeon,chang}@sookmyung.ac.kr
[2] Hankuk Aviation Universiy, Koyang, 412-791, Korea, jsahn@hau.ac.kr
[3] Hanyang University, Ansan, 426-791, Korea, doh@cse.hanyang.ac.kr

Abstract. The goal of this research is to provide an advanced programming environment for ubiquitous computing, which facilitates the development of secure and reliable ubiquitous software. A discussion follows on the design and implementation of a ubiquitous programming framework, which is based on high-level policy description language. A context-based access control manager(CACM) was implemented for context-aware access control, while an adaptation engine was integrated for context adaptation in dynamically changing environments. A simulator was also implemented for conducting experiments with this framework for ubiquitous applications.

Keywords: access control, adaptation, ubiquitous, programming environment.

1 Introduction

The effective deployment of ubiquitous software has recently become an important issue in the computing environment for assisting ubiquitous services. Numerous researchers provide software solutions for ubiquitous computing environments, comprising context-aware middlewares[4], context-based security solutions[5, 3], and programming environments for ubiquitous services[2, 8].

The goal of this research is to develop an advanced programming environment for ubiquitous computing, which facilitates the development of secure and efficient ubiquitous software. The programming environment consists of three components: a high-level policy description language, a run-time system enhanced with better context adaptation and security, and programming support tools like a simulator for simulating ubiquitous applications.

Using the policy description language[1], programmers can specify context entity relations, context-based access-control policies and context-based adaptation policies for ubiquitous application programs. A context entity relation is described by a spatial context model, which is a tree of nested entities.

Context-based access control is described by access control rules, that specify access privileges of an entity in a context. Context-based adaptation is described by adaptation rules, that specify which action to perform when a specific context condition is satisfied.

In this paper, a ubiquitous programming environment is presented, which is based on the high-level policy description language in [1]. The implementation of a context-based access-control manager (CACM) for context-aware access control is described, and an adaptation engine is integrated for context adaptation in dynamically changing environments. The implementation of a simulator is also explained as it simulates ubiquitous applications. Finally, a ubiquitous hospital application is presented to demonstrate how this system operates for context-aware access control and context adaptation in dynamically changing environments.

This paper is organized as follows. Section 2 describes the policy description language. Section 3 explains the implementation of JCAF [2], CACM, adaptation engine, and simulator. Section 4 discusses related works and Section 5 concludes with some remarks.

2 Policy Description Language

A policy specification consists of three parts: the entity relation definitions, access control rules, and adaptation rules. Declared first are the relations between the context entities to be used in the specification, then access control rules, and adaptation rules follow.

2.1 Entity Relation Definitions

A context entity in a ubiquitous environment is either a physical or a logical space, a fixed object, or a moving object. For example, consider the context of a hospital: "floor", "consulting room","sickroom", and "operating room" are space entities; "bed" is a fixed-object entity; and "PDA" is a moving-object entity. The locations of space entities and fixed entities do not change and thus are static, while those of moving objects may be changed and are dynamic. Each entity of the real world corresponds to an instance of an entity class in programs. For example, sickrooms of a hospital can be represented as a class named `Sickroom`, while an instance of the `Sickroom` class can be named `sRoom1`.

An entity relation definitions consists of two parts: context-relation and space-relation. A context-relation expresses a general relationship between entities, and a space relation expresses a space-containment relationship between a general entity and a space entity. The formal syntax for an entity relation definition is defined as follows:

$$\begin{aligned} c \in \text{Context-Relation} &::= id_1(id_2, id_3, id_4) \mid s \mid c_1, c_2 \\ s \in \text{Space-Relation} &::= id \mid id_1 : id_2 \mid id[s] \mid id_1 : id_2[s] \mid s_1 + s_2 \mid \epsilon \\ id \in \text{Identifier} & \end{aligned}$$

The type of context-relation between entities in policy specification must be defined before use. The general form of a context-relation is a quadruple of form $id_1(id_2, id_3, id_4)$, where id_1 is the kind of a relation, id_3 is the name of a relation, and id_2,id_4 are the names of entity classes(types). For example, `Location(Doctor, IsIn, Sickroom)` means that `IsIn` is the name of a relation between `Doctor` entity and `Sickroom` entity.

A physical space can be understood as a nesting hierarchical structure since it has a structure inside which other spaces are nested. The nested hierarchical structure is described as a tree that naturally defines the spatial containment relation among spaces and fixed objects. For example, the tree representation

```
Hospital[Floor[ConsultingRoom+OperatingRoom+Sickroom]]
```

indicates that: (1) there are floors in a hospital; (2) there are consulting rooms, operating rooms and sickrooms in each floor. This tree representation is an abbreviation of the following `IsIn` relations:

```
Location(Floor, IsIn, Hospital), Location(ConsultingRoom, IsIn,Floor),
Location(OperatingRoom, IsIn, Floor), Location(Sickroom, IsIn, Floor)
```

When statically determinable and necessary, the specific name of an instance can be specified along with its class name as in Fig 1. Note that an instance name must always be preceded by a class name and a colon for clarity.

2.2 Access Control Rules

An access control rule specifies that the given set of entities has the given right to the given object when the given condition is met. The following syntax is used to specify security policies:

$$\begin{aligned} p \in \text{Entity-Expression} &::= id_1 : id_2 \mid \$id \mid \$id\_n \mid * \mid p_1/p_2 \mid \ldots/p \\ r \in \text{Relation-Expression} &::= id_1(p_1, id_2, p_2) \mid \sim r \mid r_1 \wedge r_2 \\ n \in \text{Number} & \end{aligned}$$

An entity expression describes a single entity or the set of entities in the context. $id_1 : id_2$ represents an entity instance where id_2 is the name of the instance and id_1 is the name of its class, eg., `SickRoom:sRoom1`. Locational conditions can also be specified, which describe a route through the space entity hierarchy from the root. The route is called a *path expression*. For example,

```
Hospital:ubihosp/Floor:fl1/$SickRoom
```

represents any sick room in the floor named `fl1` inside the hospital named `ubihosp`. $\$id$ represents the universally quantified variable of a class named id. For example, `$SickRoom` is a universally quantified variable that can be bound to any instance of `SickRoom`. The universally quantified variables may be numbered when two or more different variables of the same class are needed. Entity expressions employ regular expression-like notation to effectively name a set of entities, and the meanings are as usual. For example, `Hospital:ubihosp/.../$Pda` means the set of all instances of `Pda` in the `ubihosp` instance of `Hospital`, and `Hospital:ubihosp/Floor:fl1/*` means the set of all instances of any entities in the `fl1` floor inside the `ubihosp` hospital.

A context relation expression describes relations between entities in context. A relation expression is interpreted as either true or false. Context relations between entities are described as a quadruple, $id_1(p_1, id_2, p_2)$, where p_1 and p_2 are entity expressions, id_1 is the kind of a relation and id_2 is the name of the relation. The quadruple means that the set of entities represented by p_1 has an id_2 relation of the id_1 relation kind with the set of entities represented by $p2$. For example, a relation "a patient has a doctor" can be expressed as:

```
HumanRelation($Patient, Has, $Doctor)
```

Technically, when this relation expression evaluates to true, the variable `$Patient` represents the set of all Patient instances having a doctor. A logical "not" operator, $\sim$, to express the negation of a relation, and a logical "and" operator, $\wedge$, is employed to express the conjunction of two relations.

The syntax of access control rules is defined as follows:

$$\begin{array}{lll} x \in \text{Access-Rule} & ::= & (p,\ o,\ r)\ \mid\ x_1\ ;\ x_2 \\ o \in \text{Object} & ::= & p.id \end{array}$$

The rule is a triple consisting of a subject, an object, and a condition. The subject is the set of entities; the object is either an entity's method name or a relation name; and the condition is a dynamic context relation which needs to be met in order for the access to be granted.

2.3 Adaptation Rules

An event happens when the change of a relation in context takes place. For example, "Dr. Kim enters the sickroom #1" is an event that sets `Location(Doctor:DrKim, IsIn, SickRoom:sRoom1)` to true. An adaptation rule specifies how to respond to events in a given context.

The syntax of adaptation rules is as follows:

$$\begin{array}{lll} d \in \text{Adaptation-Rule} & ::= & r \Rightarrow a\ \mid\ d_1\ ;\ d_2 \\ a \in \text{Action} & ::= & p_1.id(p_2)\ \mid\ id_1(p_1, id_2, p_2)\ \mid\ a_1\ ;\ a_2 \end{array}$$

An adaptation rule describes how to respond when some condition is met. The condition, r is a relation expression, and the action, a, is a method call or a new event.

2.4 An Example of Policy Specification

An example of a policy specification for a ubiquitous hospital is shown in Fig.1. First, an entity relation definition part defines the physical space of the `ubihosp` hospital and the types of context relations between entities.

This is followed by access control rules, of which Fig. 1 provides a few examples. The first rule says that the PDA of a doctor in charge of a patient has the permission to get information about the patient. The second rule permits that any doctor accompanying the doctor who is in charge of a patient can also get information about the patient with the doctor's PDA. The last rule allows only the doctor in charge of a patient to update information about the patient.

Next follow the adaptation rules. The first rule specifies that if a doctor and a patient are within a consulting room and the doctor owns a PDA, the PDA gets information about the patient. The second rule says that when a doctor approaches a patient in a sickroom, the doctor's PDA gets information about the patient. The third rule says that if a doctor assisting another doctor is in a sickroom, a new `Accompanies` relation is created between them. The last rule says that if a doctor and a patient are within an operating room and the doctor owns a PDA, the PDA gets operation information about the patient.

```
%% Entity Relation Definitions
Hospital:ubihosp[Floor[ConsultationRoom+OperatingRoom+SickRoom]]
Location(Pda, IsIn, ConsultationRoom), Location(Pda, IsIn, OperatingRoom),
Location(Pda, IsIn, SickRoom), Location(Doctor, IsIn, ConsultationRoom),
Location(Doctor, IsIn, OperatingRoom), Location(Doctor, IsIn, SickRoom),
Location(Patient, IsIn, ConsultationRoom), Location(Patient, IsIn, OperatingRoom),
Location(Patient, IsIn, SickRoom), Location(Doctor, Approaches, Patient),
HumanRelation(Patient, Has, Doctor), HumanRelation(Doctor, Assists, Doctor),
Ownership(Doctor, Owns, Pda), Behavior(Doctor, Accompanies, Doctor)

%% Access Control Rules
(Hospital:ubihosp/.../$Pda, $Patient.getInfo,
Ownership($Doctor, Owns, $Pda) ^ HumanRelation($Patient, Has, $Doctor))

($SickRoom/$Pda, $Patient.getInfo, Ownership($Doctor_1, Owns, $Pda) ^
HumanRelation($Patient, Has, $Doctor_2) ^ Behavior($Doctor_1, Accompanies, $Doctor_2))

(Hospital:ubihosp/.../$Pda, $Patient.setInfo, Ownership($Doctor,
Owns, $Pda) ^ HumanRelation($Patient, Has, $Doctor))

%% Adaptation Rules
Location($Doctor, IsIn, Hospital:ubihosp/.../$ConsultationRoom) ^
Location($Patient, IsIn, $ConsultationRoom) ^ Ownership($Doctor, Owns, $Pda)
   => $Patient.getInfo($Pda)

Location($Doctor, IsIn, Hospital:ubihosp/.../$SickRoom) ^
Location($Doctor, Approaches, $Patient) ^ Ownership($Doctor, Owns, $Pda)
   => $Patient.getInfo($Pda)

Location($Doctor_1, IsIn, Hospital:ubihosp/.../$SickRoom) ^
Location($Doctor_2,IsIn, $SickRoom) ^ HumanRelation($Doctor_1,Assists,$Doctor_2)
   => Behavior($Doctor_1, Accompanies, $Doctor_2)

Location($Doctor, IsIn, Hospital:ubihosp/.../$OperatingRoom) ^
Location($Patient, IsIn, $OperatingRoom) ^ Ownership($Doctor,
Owns,$Pda)
   => $Patient.getOperationInfo($Pda)
```

Fig. 1. An example

3 Implementation

In this section, the implementation of a context-aware access controller(CACM), which maintains the rules and controls the access, is explained. The implementation of an adaptation engine is also discussed, which executes adaptation actions when context conditions of the adaptation rules are satisfied. This implementation uses JCAF [2].

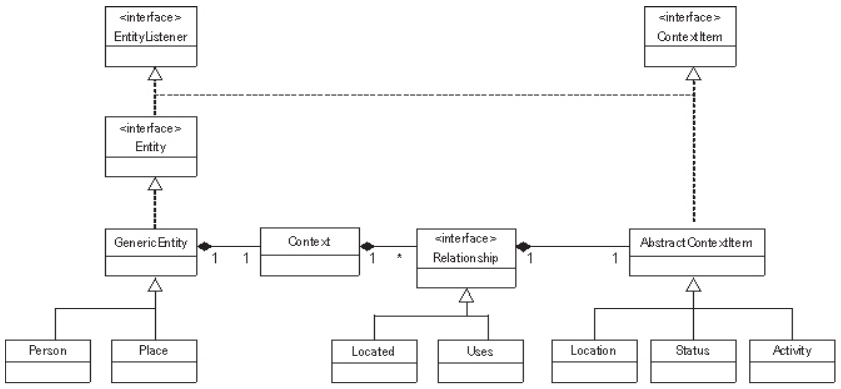


Fig. 2. UML model of an Entity with a Context in JCAF[1]

3.1 JCAF

In [2], Bardram presents Java Context-Awareness Framework(JCAF) which is a Java-based context-awareness infrastructure and API for creating context-aware applications. The core modelling interfaces provided by JCAF are `Entity`, `Context`, `Relationship`, and `ContextItem` as illustrated in Fig. 2. JCAF is also equipped with default implementations of these core interfaces. For example, the `GenericEntity` class implements the `Entity` interface and can be used to crate concrete entities through specialization. Person and Place are examples of entities. A Hospital Context and an Office Context, each knowing specific aspects about a hospital and an office, respectively, are examples of context. Physical location and the status of an operation are examples of abstract context items. Examples of relations are Located or Uses. Hence, we can model that "a Doctor is located in a Room", where a Doctor is an Entity, Located is a relation, and a Room is a context item.

The central processing part of an Entity is its `contextChanged()` method in the `EntityListener` interface. This method is guaranteed to be called by the entity container whenever this entity's context is changed. This is a very powerful way of handling context changes effectively. That is, for each possible event of context changes, an appropriate action to take can be defined for users of applications. For example, suppose that a TV's power switch should be turned on when a person approaches to the TV within a viewable distance. Then the following `contextChanged()` method can be added to the TV class:

```
public void contextChanged(ContextEvent event) {
    // If someone approaches to a TV within a viewable distance,
    // than turn the TV's power switch on.
}
```

The `contextChanged()` method is called by the JCAF runtime system as soon as the TV's context is changed. Then the `contextChanged()` method examines the event and executes an appropriate adaptation action accordingly.

3.2 Implementation of Access Control

Access control rules are managed by CACM(Context-aware Access Control Manager). Before executing any method which is under access control, ubiquitous applications check the privilege by calling the following method :

```
    checkMethodAccess(<entity1>, <entity2>, <method name>)
```

Upon the request, CACM checks whether `<entity1>` has permission to call the method of `<entity2>` or not by evaluating the condition of access control rules under the current context state. If the method

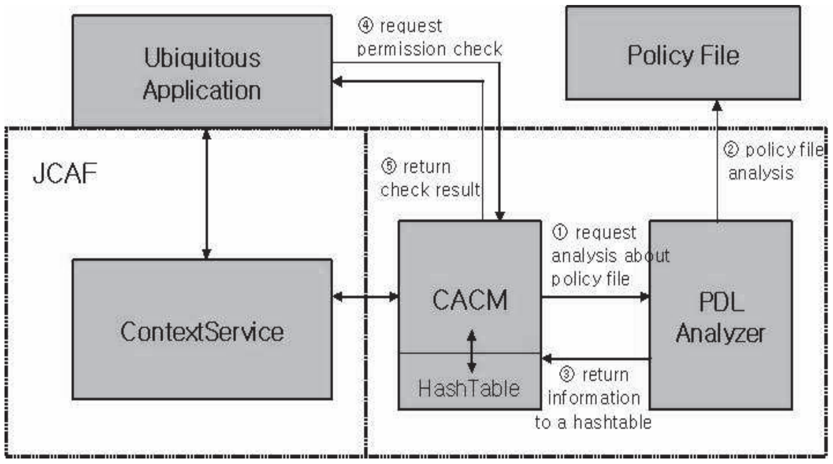
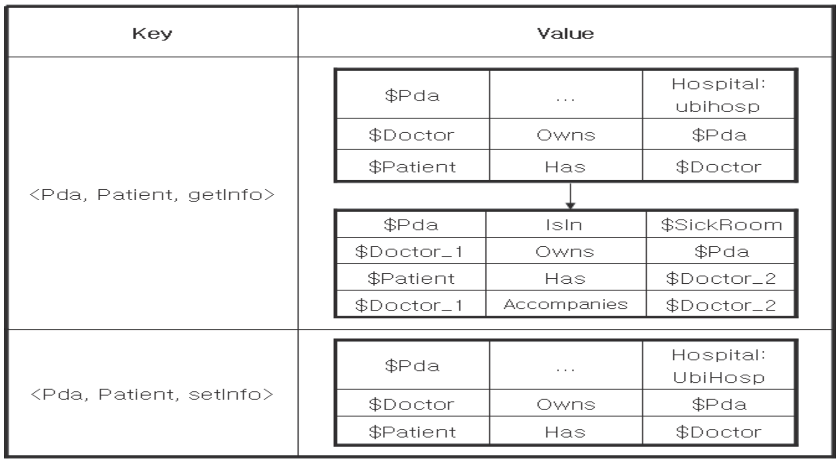


Fig. 3. The runtime architecture of CACM system



Key	Value		
<Pda, Patient, getInfo>	$Pda	...	Hospital: ubihosp
	$Doctor	Owns	$Pda
	$Patient	Has	$Doctor
	$Pda	IsIn	$SickRoom
	$Doctor_1	Owns	$Pda
	$Patient	Has	$Doctor_2
	$Doctor_1	Accompanies	$Doctor_2
<Pda, Patient, setInfo>	$Pda	...	Hospital: UbiHosp
	$Doctor	Owns	$Pda
	$Patient	Has	$Doctor

Fig. 4. The hashtable for access control of the UbiHosp application

call request is permitted by CACM, the method executes normally. Otherwise, the request is denied and `AccessControlException` is raised.

Fig. 3 shows the runtime architecture of the CACM system. CACM first requests the PDL(Policy Description Language) Analyzer to analyze the access-control rules of an input policy file. PDL Analyzer analyzes the rules and stores the conditions in a hash table and returns the table to CACM.

CACM manages a hash table which maps the 3-tuple key of the related entities and a method name to a list of context conditions for the method call. Since there can be multiple sufficient conditions for a method call, each condition becomes an element of the list. CACM examines whether there exists any condition that is satisfied under the current dynamic context. If CACM finds one, it allows the requested access. Otherwise, CACM refuses the access by rasing an exception.

Fig. 4 shows the hash table for the access control rules given in Fig. 1. The hash table stores the access control rules with keys. A key consists of the method caller, the method callee, and the method name. For example, a key `<Pda, Patient, getInfo>` means a Pda calls the method `Patient.getInfo()`. When a Pda call the method `Patient.getInfo()`, CACM searches the applicable conditions of access control rules with this key.

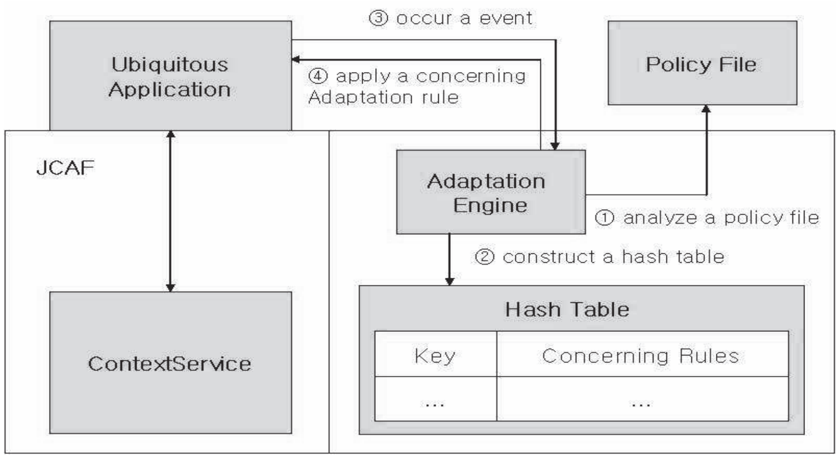


Fig. 5. Runtime architecture of adaptation engine

3.3 Implementation of Adaptation Engine

Ubiquitous applications react to dynamically changing contexts. This is implemented by an adaptation engine in this system. A user specifies the adaptation rules in a policy file, which describes how to respond when an event occurs in a given context. The adaptation engine is operated based on adaptation rules.

For example, assume there is an adaptation rule that specifies when a doctor and a patient are in the same consulting room and the doctor owns a PDA, then, in this situation, the information about the patient is displayed on the doctor's PDA automatically. Then, given any related event occurrence, such as the entrance of a patient or a doctor to a consulting room, the adaptation engine examines if all the context conditions are satisfied. If all the conditions of this rule are satisfied, the adaptation engine executes a method automatically, which displays information about the patient on the doctor's PDA.

Fig. 5 shows the system architecture for the adaptation engine. The adaptation engine is operated as follows. The adaptation engine first reads the policy file, and then analyzes the adaptation rules and stores them into a hash table.

Given a context change notification from a context service, ubiquitous programs request the adaptation engine to apply the adaptation rules by calling the following method:

```
applyAdapRules(<entity>, <relationship>, <contextItem>)
```

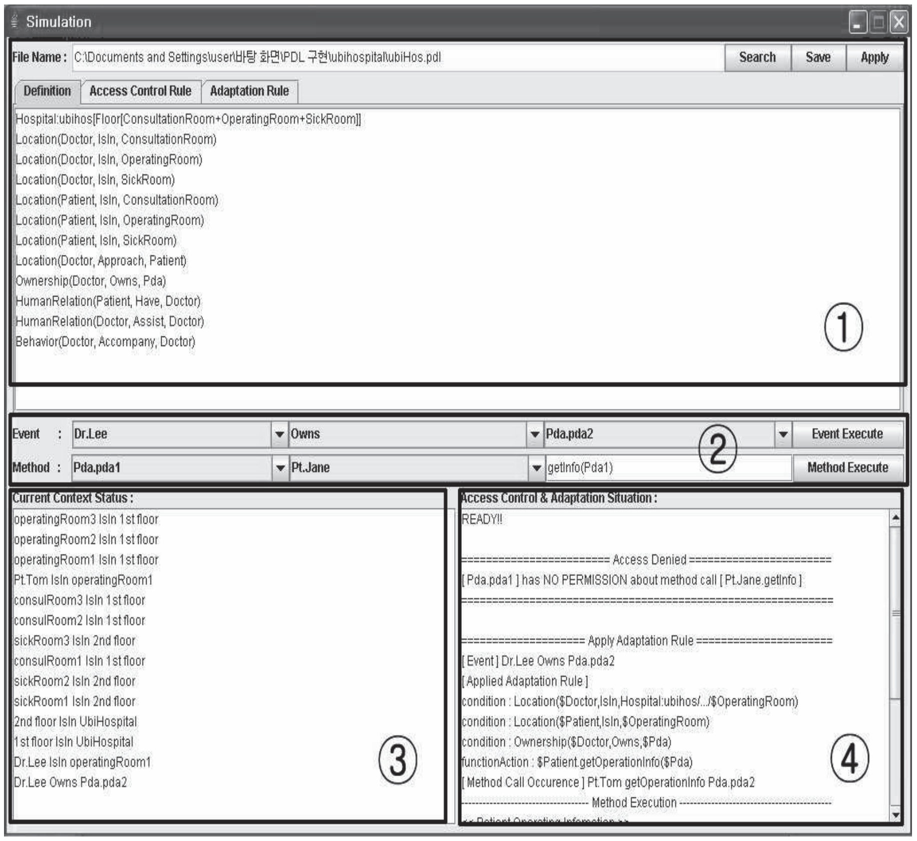
This method should be invoked inside the `contextChanged()` method that is called automatically when any context change event happens.

Upon request, the adaptation engine searches applicable adaptation rules in the hash table. If an applicable adaptation rule is found, it evaluates the conditions of the rule under the current context. If all the conditions are satisfied, then the adaptation engine applies the rule by doing its action such as calling a method or creating a new relation.

Similarly to CACM, the adaptation engine manages a hash table which maps a 3-tuple key of related entities and a relationship name to a list of context conditions for an action. Since there can be multiple related context conditions for an event, each context condition for an action becomes an element of the list.

3.4 Simulator for executing ubiquitous applications

A simulator program was developed for executing ubiquitous applications in this programming framework. A user is able to create a policy file for a ubiquitous application. Then the user can execute the ubiquitous application. With this simulator, users can test applications with various scenarios of access control policies and adaptation policies.



```
File Name : C:\Documents and Settings\user\바탕 화면\PDL 구현\ubihospital\ubiHos.pdl

Hospital:ubihos[Floor[ConsultationRoom+OperatingRoom+SickRoom]]
Location(Doctor, IsIn, ConsultationRoom)
Location(Doctor, IsIn, OperatingRoom)
Location(Doctor, IsIn, SickRoom)
Location(Patient, IsIn, ConsultationRoom)
Location(Patient, IsIn, OperatingRoom)
Location(Patient, IsIn, SickRoom)
Location(Doctor, Approach, Patient)
Ownership(Doctor, Owns, Pda)
HumanRelation(Patient, Have, Doctor)
HumanRelation(Doctor, Assist, Doctor)
Behavior(Doctor, Accompany, Doctor)

Event  :  Dr.Lee     Owns      Pda.pda2
Method :  Pda.pda1   Pt.Jane   getInfo(Pda1)
```

```
Current Context Status :
operatingRoom3 IsIn 1st floor
operatingRoom2 IsIn 1st floor
operatingRoom1 IsIn 1st floor
Pt.Tom IsIn operatingRoom1
consulRoom3 IsIn 1st floor
consulRoom2 IsIn 1st floor
sickRoom3 IsIn 2nd floor
consulRoom1 IsIn 1st floor
sickRoom2 IsIn 2nd floor
sickRoom1 IsIn 2nd floor
2nd floor IsIn UbiHospital
1st floor IsIn UbiHospital
Dr.Lee IsIn operatingRoom1
Dr.Lee Owns Pda.pda2
```

```
Access Control & Adaptation Situation :
READY!!

======================= Access Denied =======================
[ Pda.pda1 ] has NO PERMISSION about method call [ Pt.Jane.getInfo ]
=============================================================


=================== Apply Adaptation Rule ======================
[ Event ] Dr.Lee Owns Pda.pda2
[ Applied Adaptation Rule ]
condition : Location($Doctor,IsIn,Hospital:ubihos/.../$OperatingRoom)
condition : Location($Patient,IsIn,$OperatingRoom)
condition : Ownership($Doctor,Owns,$Pda)
functionAction : $Patient.getOperationInfo($Pda)
[ Method Call Occurence ] Pt.Tom getOperationInfo Pda.pda2
---------------------------------- Method Execution ------------------------------------------
```

Fig. 6. The simulator for ubiqutous applications

Fig. 6 shows a snapshot of the simulator, which displays execution traces visually with current context information. Each numbered part has the following role: In part 1, a user selects a policy file for an application. The policy information is to be displayed into three separate regions: context definitions, access control rules and adaptation rules. A user is also able to modify a policy file and apply the new version repeatedly. Part 2 lists possible events to be generated and possible methods to be executed. This information is gathered by analyzing selected policy files. If a user selects events or methods in a sequence, then a scenario is made by this selected sequence. Part 3 displays the current context. The current context consists of the relations of `Entity` and `ContextItem`. Part 4 displays the result of the program execution. If a method call is denied by the access control, the part 4 displays an alarm message with the explanation. If an event is generated automatically by the adaptation rules, part 4 displays a message about the execution result and the applied adaptation rule.

4 Related Works

Several groups have conducted research to provide software solutions for the ubiquitous computing environment, which includes the programming framework, context-aware middleware, and context-based security for ubiquitous service.

JCAF developed by Badram is a framework for ubiquitous computing that uses Java [2]. This enables simply and clearly defined context. This implementation for access control and adaptation is based on the JCAF framework.

A. Ranganathan and Roy H. Campbell developed a context-aware system called Gaia [7]. This system gathers context information, according to the kind of relation, and it is analyzed and disposed of using first-order logic. This system reacts to context circumstance with the content defined in a configuration file. However, in this configuration file, only instance values can be used.

Bellavista et. al proposed a middleware for context-aware resource management, called CARMEN, capable of supporting the automatic reconfiguration of wireless Internet services in response to context changes without any intervention on the service logic [4]. CARMEN determines the context on the basis of metadata, which include declarative management policies and profiles for user preferences, terminal capabilities, and resource characteristics.

5 Concluding Remarks

A ubiquitous programming framework has been developed that is based on high-level policy description language. A context-based access control manager for context-aware access control has been implemented, and an adaptation engine for context adaptation in dynamically changing environments has been integrated. A simulator for ubiquitous applications was also implemented.

The final goal of this research is to develop an advanced programming environment for ubiquitous computing, which facilitates the development of secure and adaptive ubiquitous software. A type of system for the specification language will be designed, while the consistency policy specification will automatically be checked based on the type of system. Ubiquitous programming support tools will be developed for program analysis and monitoring. The program analysis tools help programmers develop secure and efficient programs that are based on static analysis such as context-flow analysis, access control analysis, secure information-flow analysis, and exception analysis. The program monitoring tools allow programmers to examine the execution traces of programs such as context change, exception handling, and access control during execution.

Acknowledgements

This work was supported by grant No. R01-2006-000-10926-0 from the Basic Research Program of the Korea Science and Engineering Foundation

References

1. Joonseon Ahn, Byeong-Mo Chang, and Kyung-Goo Doh, A Policy Description Language for Context-based Access Control and Adaptation in Ubiquitous Environment, TRUST06, August, 2006
2. J. E. Bardram, The Java Context Awareness Framework-A Service Infrastructure and Programming Framework for Context-Aware Applications, Third International Conference, Pervasive2005, Munich, Germany, May, 2005.
3. E. Cho and K. Lee, Security Checks in Programming Languages for Ubiquitous Environments, *Proceedings of 2004 Workshop on Pervasive, Security, Privacy and Trust*, Aug. 2004.
4. P. Bellavista, A. Corradi, R. Montanari, Context-Aware Middleware for Resource Management in the Wireless Internet, *IEEE Transactions on Software Engineering* , Vol. 29, No. 12, December 2003.
5. Antonio Corradi, Rebecca Montanari, Daniela Tibaldi, Context-based Access Control for Ubiquitous Service Provisioning, *Proceedings of the 28th International Computer Software and Applications Conference(COMPSAC'04)*, 2004.
6. A. Ranganathan, Roy H. Campbell, An infrastructure for context-awareness based on first order logic, Springer-Verlag London Limited 2003, November 2003
7. M. Roman, C.K. Hess, R. Cerqueira, A. Ranganat, R.H. Campbell, K. Nahrstedt, Gaia: A Middleware Infrastructure to Enable Active Spaces. IEEE Pervasive Computing, pp. 74-83, 2002
8. D. J. Scott, Abstracting application-level security policy for ubiquitous computing, University of Cambridge, Computer Laboratory, Technical Report UCAM-CL-TR-613, January 2005.
9. D. Wichadakul, X. Gu and K. Nahrstedt, A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications, *Proceedings of Multimedia'02*, December, 2002, Juan-les-Pins, France.