유비쿼터스 컴퓨팅을 위한 접근제어와 상황적응 시스템*

이지연[O1] 오민경[1] 창병모[1] 안준선[2] 도경구[3]
[1]숙명여자대학교 {jiyeon[O], omk, chang}@sookmyung.ac.kr
[2]한국항공대학교 jsahn@hau.ac.kr
[3]한양대학교 doh@cse.hanyang.ac.kr

Access Control and Adaptation System for Ubiquitous Computing

Jiyeon Lee[O1], Minkyoung Oh[1], Byeong-Mo Chang[1], Joonseon Ahn[2], Kyung-Goo Doh[3]
[1]Sookmyung Women's University, Seoul
[2]Hankuk Aviation University, Koyang
[3]Hanyang University, Ansan

요 약

본 연구는 신뢰성 있는 유비쿼터스 프로그램을 개발하기 위한 보다 효율적인 프로그래밍 환경을 제공함을 목적으로 한다. 이 프로그래밍 환경은 정책 기술 언어와 이를 바탕으로 접근 제어와 상황 적응을 관리하는 실행시스템을 제공한다. 프로그래머는 정책 기술 언어를 통해 메소드에 대한 접근 권한과 변화되는 상황에 적응하는 규칙들을 쉽게 표현할 수 있다. 본 논문의 실행 시스템은 정책 파일의 규칙들에 따라 객체가 메소드에 대한 접근 권한이 있는지를 검사하는 접근 제어기(access controller)와 변화되는 상황에 적응하여 자동으로 반응하도록 해주는 적응 엔진(adaptation engine)의 형태로 구현되었다. 또한 이 시스템을 이용하여 개발된 유비쿼터스 프로그램을 시뮬레이션 할 수 있는 시뮬레이터를 제공한다.

1. 서 론

유비쿼터스 컴퓨팅을 위한 많은 장치들의 발전과 더불어 유비쿼터스 서비스를 위한 소프트웨어 기술들도 더욱 강조되고 있다. 따라서 효과적이고 안전한 유비쿼터스 프로그래밍 환경은 매우 중요하다고 할 수 있다.

본 연구에서는 유비쿼터스 프로그램을 효과적으로 개발하기 위한 프로그래밍 환경을 제공함을 목적으로 한다. 이 환경은 정책 기술 언어(Policy Description Language)[4]와 실행 시스템으로 구성된다. 프로그래머는 작성할 유비쿼터스 프로그램의 접근 권한과 상황 적응 규칙을 정책 기술 언어를 통해 쉽게 표현할 수 있으며 실행 시스템은 프로그램이 실행될 때 기술된 규칙에 따라 접근을 제어하고 변화되는 상황에 자동적으로 적응하도록 해준다.

본 논문에서는 접근제어와 상황 적응을 위한 실행 시스템과 개발된 프로그램을 시뮬레이션 할 수 있는 시뮬레이터를 구현하였다. 이 실행 시스템은 Java 기반 상황 인식 프로그래밍 프레임워크인 JCAF(Java Context-Awareness Framework)[1]를 바탕으로 객체의 메소드 접근을 제어하는 접근 제어기(access controller)와 변화하는 상황에 유연하게 적응하도록 하는 적응 엔진(adaptation engine) 형태로 구현하였다.

이 실행 시스템은 사용자가 정의한 정책 파일을 분석하여 효과적으로 메소드 접근을 제어하고 프로그램이 상황 변화에 따라 유연하게 적응하도록 한다. 이를 이용해 프로그래머는 보다 안전하고 효과적인 유비쿼터스 프로그램을 구현할 수 있다.

* 본 연구는 한국과학재단 특정기초연구 (R01-2006-000-10926-0) 지원으로 수행되었음.

이 논문은 정책 기술 언어, 시스템 구현, 관련 연구, 결론 및 향후 연구로 구성된다.

2. 정책 기술 언어

정책 기술 언어는 엔티티 릴레이션 정의, 접근 제어 규칙, 상황 적응 규칙의 세 부분으로 구성된다. 먼저 정책 기술에서 사용될 엔티티와 그들 간의 관계에 대해서 정의를 하고, 다음으로 접근 제어 규칙과 상황 적응 규칙을 기술한다.

2.1 엔티티 릴레이션 정의

유비쿼터스 환경에서 엔티티는 물리적・논리적인 공간이나, 움직이거나 움직이지 않는 객체를 나타낸다. 예를 들어, “floor", "sickroom"은 공간 엔티티이고, “PDA"는 움직이는 엔티티, “bed"는 움직이지 않는 엔티티이다. 공간 엔티티나 움직이지 않는 엔티티의 경우는 그 위치가 고정되어 변하지 않는 반면, 움직이는 엔티티의 경우에는 동적으로 그 위치나 상황이 변하게 된다. 실세계의 엔티티들은 프로그램에서 *Entity* 클래스의 인스턴스로서 표현된다.

엔티티 릴레이션 정의부는 Context-Relation과 Space-Relation으로 구성된다. Context-Relation는 일반적인 두 엔티티들 사이의 관계을 표현하고, Space-Relation은 일반 엔티티와 공간 엔티티간의 공간적인 포함 관계를 표현한다.

엔티티 릴레이션 정의의 문법은 다음과 같다.

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

Context-Relation은 $id_1(id_2,id_3,id_4)$의 구조를 갖는다. 이때 id_1은 릴레이션의 종류를 나타내고, id_2는 주체가 되는 엔티티 이름, id_3는 릴레이션 이름, id_4는 대상이 되는 엔티티 이름을 나타낸다. 예를 들어, Location(Doctor, IsIn, SickRoom)은 Doctor가 SickRoom과 IsIn 릴레이션을 맺을 수 있음을 의미한다. 정책 파일에 Context-Relation을 적음으로써 엔티티들간의 릴레이션을 더욱 명확히 하고, 응용 프로그램 상에서 쓰여질 릴레이션들을 예상할 수 있도록 도와준다. 따라서 정책 기술에서 사용될 릴레이션들은 반드시 미리 정의되어야만 한다.

공간이라는 것은 항상 다른 공간 안에 포함되어 있기 때문에, 이를 표현하기 위해 포개어진 계층 구조를 사용할 수 있다. Space-Relation은 계층 구조로 공간이나 움직이지 않는 객체의 상황을 표현한다. 중괄호는 이러한 계층구조를 표현하기 위함이다. 이렇게 표현된 공간 계층 구조는 IsIn관계가 생략되어 표현된 것이라고 생각될 수 있다. 예를 들어, Hospital:ubihosp[Floor[ConsultationRoom+OperatingRoom+SickRoom]]이라는 릴레이션 정의가 있다면 이것은 ubihosp라는 병원이 층으로 이루어져 있고, 각 층은 진료실, 수술실, 입원실을 포함하고 있음을 의미한다. 명확히 결정된 인스턴스를 사용해야 할 필요가 있을 때, 위의 예에서처럼, 엔티티의 클래스 이름 뒤에 콜론을 찍고 인스턴스 이름을 적을 수 있다.

2.2 접근 제어 규칙 기술

접근 제어 규칙은 어떠한 조건을 만족할 때, 엔티티 집합들이 메소드에 접근 권한이 있는지에 대한 내용을 기술한다.

우리는 정책들을 기술하는데 다음의 식을 이용한다.

```
p ∈ Entity-Expression := id1:id2 | $id | $id_n | *
                       | p1/p2 | .../p
r ∈ Relation-Expression := id1(p1,id2,p2) | ~r | r1 ^ r2
n ∈ Number
```

Entity-Expression은 어떤 한 개의 엔티티나 엔티티의 집합을 표현한다. id_1:id_2는 Pda:BobPda와 같이 id_1이라는 이름을 갖는 클래스의 인스턴스 id_2를 표현한다. $id는 id라는 이름을 갖는 클래스의 일반적인 인스턴스 중의 하나를 정하는 변수를 의미한다. $Room은 Room이라는 클래스의 어떤 인스턴스를 담는 변수가 된다.

그리고 공간 엔티티들간의 종속 관계 표현을 위해, '/'를 사용하게 된다. 예를 들어,

Hospital:ubihosp/Floor:fl1/$SickRoom

은 ubihosp라는 병원에 fl1이라는 층에 있는 어떤 병실을 의미하게 된다. 또한 엔티티의 표현을 효율적으로 하기위한 여러 가지 문법들을 제공하고 있는데, 예를 들어,

Hospital:ubihosp/.../$Pda

은 ubihosp에 있는 모든 Pda들의 인스턴스를 의미하게 되고

Hospital:ubihosp/Floor:fl1/*

은 ubihosp 병원의 fl1층에 있는 모든 인스턴스를 표현하게 된다.

다음으로, Relation-Expression은 엔티티들 간의 관계에 대해서 표현하며, 이것은 참, 거짓으로 해석될 수 있다. 예를 들어, HumanRelation($Patient, Has, $Doctor)은 Patient라는 클래스의 한 인스턴스가 어떤 Doctor를 주치의로 Has하는 관계를 나타낸다. '~'은 "not"연산자를 의미하며, '^'은 "and"연산자를 대신한다.

접근 제어 정책 기술의 문법은 다음과 같다.

```
x ∈ Access-Rule ::= (p,o,r) | x1 x2
o ∈ object ::= p.id
```

Access-Rule은 주체(p)가 조건(r)을 만족할 때 대상(o)를 호출할 수 있다는 정책을 정의한다. 메소드를 호출하는 호출자가 주체를 의미하며 이는 엔티티의 집합으로 표현된다. 또한 대상은 엔티티의 메소드 이름으로 표현되며, 권한을 갖기 위해 필요한 조건은 Context-Relation으로 표현된다. 2.4절의 예제를 통해 Access-Rule의 이해를 도울 수 있다.

2.3 상황 적응 규칙 기술

실행 시간 중 상황에 변화가 생기게 되면 이벤트가 발생한다. 상황 적응 정책에서는 주어진 상황에서 이벤트가 발생하였을 때, 어떻게 반응해야 하는지에 대한 내용을 기술한다.

상황 적응 정책 기술의 문법은 다음과 같다.

```
d ∈ Adaptation-Rule ::= r => a | d1 d2
a ∈ Action ::= p1.id(p2) | id1(p1,id2,p2) | a1;a2
```

Adaptation-Rule은 조건과 행동 두 가지를 묘사한다. 조건(r)과 같은 상황을 만나면 그에 적응하는 행동(a)이 발생되도록 한다는 의미이다. 적응 규칙이 실행되기 위한 조건은 Context-Relation으로 표현되고, 이 조건이 만족되었을 때 자동으로 실행되는 행동은 이벤트 발생이나 메소드 호출로 표현된다. Adaptation-Rule에 대해서는 2.4절의 예제에서 자세히 설명한다.

2.4 정책 기술 언어를 이용한 정책 파일의 예

그림 1.은 유비쿼터스 병원에 대한 정책 파일의 예이다.

우선 엔티티 릴레이션의 정의부에는 ubihosp의 물리적인 공간정보와 엔티티들간의 상황 릴레이션 정보들이 정의되어 있다.

다음으로 접근 제어 규칙 정의부는 유비쿼터스 병원 프로그램에 적용될 엔티티들의 권한에 대한 정의이다. 그림 1.의 접근제어 정책에 대하여 설명을 하면 다음과 같다. 첫번째 규칙은 주치의인 의사가 자신의 Pda를 통해 환자의 정보를 볼 권한을 갖는다는 규칙이다. 두 번째 규칙은 어떤 의사가 병실에 들어와서 환자의 정보를 보려고 할 때, 주치의인 의사를 동반한 경우라면 권한을 갖는다는 규칙이다. 마지막 규칙은 주치의가 환자에 대해 정보를 수정할 권한을 갖는다는 의미이다.

```
%% 엔티티 릴레이션 정의
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),
 Ownership(Doctor, Owns, Pda),
 HumanRelation(Patient, Has, Doctor),
 HumanRelation(Doctor, Assists, Doctor),
 Behavior(Doctor, Accompanies, Doctor)

%% 접근 제어 규칙
(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))

%% 상황 적응 규칙
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)
```

그림 1. 유비쿼터스 병원의 정책 파일

그 다음, 상황 적응 규칙 정의부는 유비쿼터스 병원이 상황의 변화에 대해 자동으로 반응하도록 규칙을 정한 것이다. 예를 들어, 첫 번째 규칙은 의사가 있는 진료실에 환자가 들어오면 의사의 Pda에 자동으로 환자의 정보를 보여주도록 하는 규칙이다. 두 번째 규칙은 의사가 입원실에 들어가서 환자에게 다가가면 해당 환자의 정보를 자동으로 의사의 Pda에 보여주도록 하는 규칙이다. 세 번째 규칙은 의사가 Assists관계에 있는 의사와 함께 병실에 들어오면 Accompanies관계를 생성해주는 규칙이고, 마지막 규칙은 의사가 수술실에 들어오면 자동으로 환자의 수술정보를 Pda에 보여주도록 하는 규칙이다.

3. 시스템 구현

이 시스템은 상황 인식 프로그래밍 프레임워크인 JCAF[1](Java Context-Awareness Framework)를 바탕으로 구현되었고, CACM(Context-based Access Control Manager)이라 부르는 접근 제어기와 적응 엔진으로 구성된다. 프로그래머가 앞서 언급한 정책 기술 언어를 통해 정책 파일을 작성하여 유비쿼터스 프로그램과 함께 실행시키면 CACM과 적응 엔진은 작성된 정책 파일을 분석하여 해시테이블 형태로 저장한다. 그리고 실행되는 과정에서 프로그램이 메소드 호출에 대해 권한이 있는지 검사를 요청하면 CACM은 해시테이블의 정보와 비교하여 권한 검사 결과를 반환한다. 또한 적응 엔진은 실행 중 해시테이블에 저장된 규칙과 같은 상황(context)이 발생하면 규칙을 적용하여 새로운 이벤트를 발생시킨다.

3.1 JCAF

JCAF[1]는 상황 인식 프로그램 개발을 위한 하부 구조와 API를 제공하는 Java기반의 프레임워크이다. JCAF는 ContextService를 통해 여러 Entity들에 대한 정보 수집과 관리를 하게 된다.

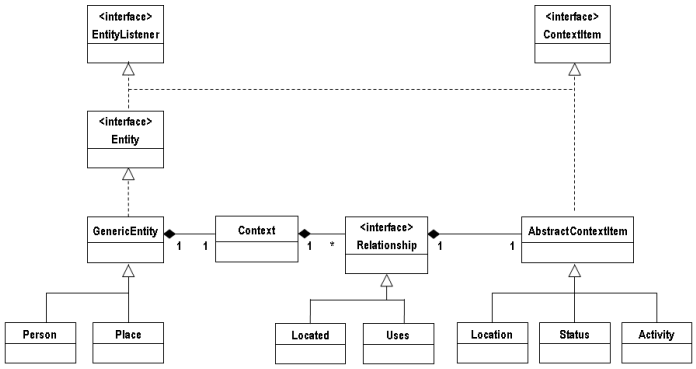

그림 2. JCAF의 핵심 요소들

JCAF가 가지는 가장 핵심적인 요소는 Entity, Context, Relationship, ContextItem이다. 위의 그림 2.는 이러한 요소들의 관계를 보여준다. 각각의 Entity는 하나의 Context를 가지게 되고, 각 Entity가 다른 item들과 맺고 있는 Relationship들의 집합이 Context를 구성하게 된다. Entity의 예로는 사람, 장소, 사물 등이 될 수 있고, ContextItem의 예로는 위치, 상태 등이 될 수 있다. 예를 들어 PDA가 로비에 있다면 PDA는 Entity, located는 Relationship, lobby는 ContextItem이 된다.

Entity를 처리하는 주요 부분은 EntityListener 인터페이스에 있는 contextChanged() 메소드이다. 이 메소드는 Entity의 Context가 바뀔 때마다 Entity Container에 의해 불릴 것이라는 사실이 보장된다. 이것은 Context 변화를 효과적으로 다루

는 매우 강력한 방법이다. 즉, 상황 변화에 따라 발생 가능한 이벤트들을 처리하기 위해 취해질 수 있는 행동들이 유비쿼터스 프로그램 개발자에 의해 정의될 수 있다.

```
public void contextChanged(ContextEvent event) {
   // 이벤트 발생 시에 적용시키고 싶은 액션을 기술한다.
}
```

3.2 접근 제어의 구현

접근 제어 정책은 실행 시간에 ContextService와 함께 구동되는 CACM(Context-aware Access Control Manager)에 의해 관리된다. CACM은 권한 검사를 위한 다음의 메소드를 제공하고 있다. 접근 제어를 위하여 메소드 실행 전에 아래의 메소드를 호출하여 <entity1>이 <entity2>의 메소드 <method name>을 접근할 권한을 갖는지 검사한다.

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

권한 검사가 필요한 부분에 checkMethodAccess(A)를 호출하면 CACM은 현재의 상황(context)과 정책 파일이 저장된 해시테이블을 비교하여 결과를 참, 거짓으로 반환한다. 참이 반환되면 아무런 제한 없이 메소드가 실행되고 거짓이 반환되면 AccessControlException()이 발생한다.

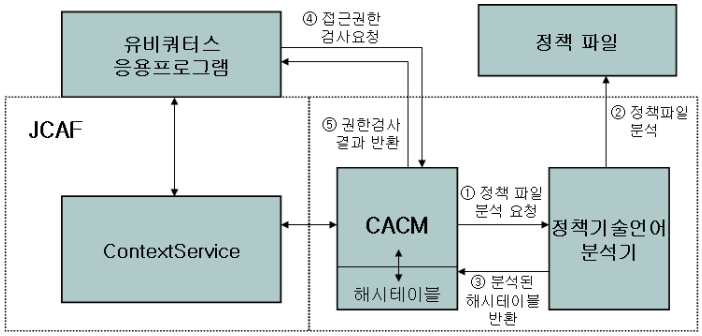

그림 3. CACM의 실행 구조

그림 3.은 CACM이 실행되는 시스템의 구조를 보여준다. 접근 제어가 이루어지는 과정은 다음과 같다. 우선 프로그래머는 앞서 설명된 정책 기술 언어를 통해 엔티티들이 메소드에 대해 갖는 권한을 정책 파일에 기술한다. 그 다음 CACM은 정책 기술 언어 분석기에게 정책 파일을 분석해 줄 것을 요청한다. 정책 기술 언어 분석기는 정책 파일을 분석하여 해시테이블 형태로 정보를 저장하고, 생성된 해시 테이블을 다시 CACM으로 반환한다. 그리고 유비쿼터스 프로그램은 실행 중 메소드 호출이 발생할 때 어떠한 엔티티 객체가 호출에 대한 권한을 갖는지 CACM에게 권한 검사를 요청하게 된다. 마지막으로 CACM은 현재의 상황(context)과 해시테이블의 조건이 일치하는지 분석하여 결과를 반환한다.

정책 파일을 저장하고 있는 해시 테이블은 메소드의 호출자인 엔티티 객체와 메소드를 가지고 있는 피호출자, 그리고 메소드 이름을 키로 갖는다. 예를 들어 <Pda, Printer, print>를 키로 하면 어떤 Pda가 프린터의 print()메소드를 호출할 때 필요한 권한과 조건들이 이 키의 값으로 저장되어 있게 된다. 하나의 키로 접근할 수 있는 조건의 집합들은 여러 개일 수 있다. 조건 집합은 하나의 접근 제어 규칙에 의해 얻어지는 조건들의 집합이다. 키를 통해 얻어진 여러 조건 집합들 중 하나의 조건 집합이라도 만족하게 되면 CACM은 참을 반환하고 검사를 마치게 된다. 하지만 모든 조건 집합들을 다 만족하지 않으면 거짓을 반환한다.

3.3 적응 엔진의 구현

상황 적응 정책은 실행 시간에 ContextService와 함께 구동되는 적응 엔진(Adaptation Engine)에 의해 관리된다.

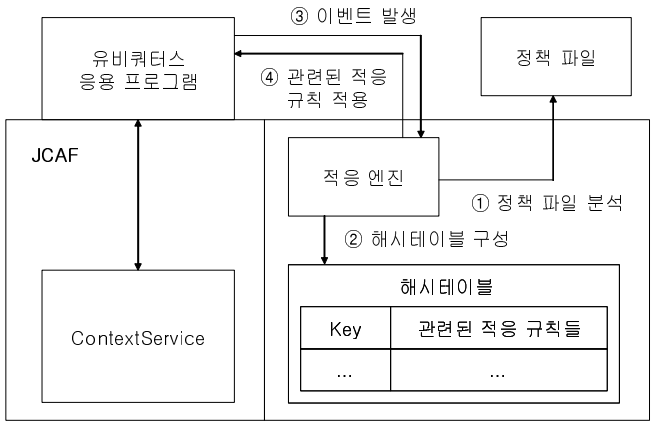

그림 4. 적응 엔진의 실행 구조

그림 4.는 적응 엔진이 실행되는 시스템의 구조를 보여준다. 적응 엔진은 다음과 같은 순서로 실행된다. 먼저 적응 엔진은 프로그래머가 작성한 정책파일을 분석하고, 그 정보를 이용하여 해시테이블을 구성한다. 그 후, 응용 프로그램에서의 상황 변화가 인지되면, 해시테이블에서 관련된 적응 규칙이 있는지 찾아보게 된다. 만약 관련된 적응 규칙이 발견되면, 현재의 상황을 분석하여 적응 규칙의 조건과 일치하는지 검사하고, 조건의 일치여부에 따라 그에 해당하는 행동을 자동적으로 수행하게 된다.

적응 엔진은 응용 프로그램에서의 상황 변화를 인지하기 위하여 다음과 같은 메소드를 제공하고 있다.

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

이벤트가 발생하였을 때 자동으로 호출되는 contextChanged() 메소드에서 이 메소드를 호출하면, 적응 엔진은 자동적으로 상황 변화를 인지할 수 있게 된다. 이 메소드 호출 시, 변화된 관계에 대한 정보가 적응 엔진에게 넘겨지게 되고, 적응 엔진은 이 정보를 분석하여 상황에 맞는 적절한 반응을 보이게 된다.

적응 엔진에서는 어떤 상황 변화가 일어났을 때 관련된 적응 규칙을 빨리 찾기 위해서 해시테이블을 사용하고 있다. 이때 해시테이블은 이벤트의 주요 정보들로 키를 구성하고, 그와 관

련된 규칙들을 값으로 저장한다. 만약 한 이벤트에 관련된 규칙들이 많다면 하나의 키에 대해 여러 개의 규칙들이 값으로 저장될 것이다. 실제 실행 중 적응 엔진이 상황 변화를 인식하게 되면, 그 상황 변화를 일으킨 이벤트의 정보를 분석하여 키를 만들고 그것을 이용하여 해시테이블을 검색한다. 예를 들어, 어떤 환자가 진료실에 들어오면 자동으로 의사의 PDA에 환자의 정보를 보여 주는 규칙이 있다고 하자. 그러면 해시테이블은 <Patient, IsIn, ConsultationRoom>을 키로 하여, 이 규칙을 저장하게 된다. 그리고 적응 엔진은 실제로 환자가 진료실에 들어왔다는 이벤트가 발생했을 때, 이와 똑같은 키를 만들어 해시테이블을 검색하게 된다.

3.4 응용 프로그램 실행을 위한 시뮬레이터

우리는 본 시스템을 이용하여 구현한 유비쿼터스 프로그램의 실행을 위하여 시뮬레이터를 개발하였다. 이 시뮬레이터를 이용하면 사용자는 쉽게 수많은 시나리오로 접근 제어와 상황 적응을 시험해 볼 수 있다.

사용자는 우선 이 시뮬레이터를 실행시킨 뒤, 적용할 정책 파일을 선택한다. 그리고 발생할 수 있는 상황 릴레이션이나 메소드 호출을 선택하면 이벤트를 발생시켜 그에 대한 해당 결과를 보여준다. 또한 이 시뮬레이터는 현재 상황을 한 눈에 볼 수 있도록 구성되어 있고, 정책 파일을 수정해서 다시 적용해 볼 수 있는 기능을 가지고 있다.

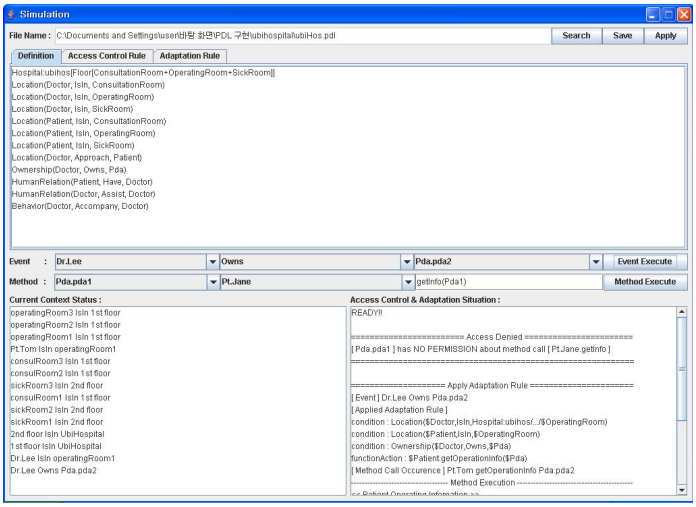

그림 5. 시뮬레이션 프로그램

4. 관련 연구

유비쿼터스 컴퓨팅 환경에 대한 몇몇 연구들이 진행되어 왔다.

Bardram이 개발한 JCAF[1]는 Java 기반 상황 인식 프로그래밍 프레임워크이다. 이것은 실세계의 상황 정보를 효과적으로 모아서 관리할 수 있도록 하는 기반 구조와 프로그래밍 인터페이스를 제공한다. 본 연구는 이 JCAF를 바탕으로 하여 접근 제어와 상황 적응을 보다 효과적으로 수행하는 시스템을 구현하였다.

A. Ranganathan과 Roy H. Campbell은 Gaia를 기반으로 상황 인식 시스템[2]을 개발하였다. 이 시스템에서는 다양한 상황 정보를 종류별로 모아서 받아들이고, 그 정보를 1차 논리 구조(fist-order logic)에 기반하여 분석하고 처리한다. 먼저 상황 인식 규칙을 정의하고 이에 따라 시스템이 적절히 반응하도록 되어 있다. 그러나 이 시스템은 규칙을 정의할 때 실제 인스턴스 값만을 이용해야 한다.

Bellavista와 Montanari가 개발한 CARMAN[3]은 상황 인식 자원 관리를 위한 미들웨어이다. CARMAN은 메타데이터를 기본으로 하여 무선인터넷 환경에서 어떤 로직의 개입도 없이 자동적으로 변화된 환경을 재구성한다. CARMAN의 메타데이터는 객체들의 특징을 정의한 Profile과 이것들을 어떻게 다루어야 할지를 정의한 Policy로 구성된다.

5. 결론 및 향후 연구

본 시스템은 안전하고 상황에 적응하는 유비쿼터스 프로그래밍을 지원하는 효과적인 환경을 제공하고자 하였다. 사용자는 본 연구에서 개발된 정책 기술 언어를 통해 쉽고 자연스럽게 접근 제어와 적응 규칙을 표현할 수 있고 시스템은 이를 분석하여 효율적으로 자원 접근을 제어하고 상황의 변화에 적응하는 유연한 프로그램을 만들어 준다. 또한 시뮬레이터를 통하여 가상으로 애플리케이션을 실행시켜 볼 수 있다.

향후, 우리는 유비쿼터스 프로그램의 분석과 모니터링을 위한 툴을 개발할 것이다. 이 툴은 사용자에게 더 정확하고 효율적인 프로그램을 개발할 수 있도록 도울 것이다. 또한 정책언어 기술을 위한 타입 검사 시스템과 정책의 일관성 검사 시스템을 개발할 것이다.

[참고문헌]

1. 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.

2. A. Ranganathan, Roy H. Campbell, An infrastructure for context-awareness based on first order logic, Springer-Verlag London Limited 2003, November 2003

3. Paolo Bellavista, Rebecca Montanari, Context-Aware Middleware for Resource Management in the Wireless Internet, reference IEEECS, Aug.2003

4. 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