유비쿼터스 컴퓨팅을 위한 상황 적응 시스템

오 민 경[†] · 창 병 모[††]

요 약

유비쿼터스 컴퓨팅 환경은 변화하는 상황에 적응함으로써 사람들에게 더 좋은 서비스를 제공할 수 있다. 본 논문에서는 유비쿼터스 프로그램을 정해진 규칙에 따라 변화하는 상황에 적응시키는 상황 적응 시스템을 개발하였다. 본 시스템은 개발자로 하여금 상황 적응 정책만을 간단히 기술하게 함으로써 프로그램을 변화하는 상황에 적응시킬 수 있도록 해준다. 본 시스템의 상황 적응 엔진은 상황 적응 젇책에 따라 유비쿼터스 프로그램을 변화하는 상황에 적응시킨다. 또한 본 시스템은 개발자로 하여금 가상으로 자신의 유비쿼터스 프로그램을 변화하는 상황에 적응시켜 볼 수 있도록 하기 위하여 시뮬레이터를 제공한다. 본 시스템은 자바 기반 상황 인식 프로그래밍 프레임워크인 JCAF 위에서 구현되었다.

키워드 : 상황 인식 컴퓨팅, 상황 적응 시스템, 상황 적응 정책, JCAF

A Context Adaptation System for Ubiquitous Computing

Minkyung Oh[†] · Byeong-Mo Chang[††]

ABSTRACT

The ubiquitous computing environment could provide better service to users by adapting to changing contexts. In this paper, we developed a context adaptation system, which enables an ubiquitous program to adapt to different contexts, following its adaptation rules. Using this system, programmers can develop ubiquitous programs suitable for changing contexts, by describing the context adaptation policy. The context adaptation engine of this system fits the ubiquitous program to the current context based on the context adaptation rules. This system was implemented using JCAF, context-aware programing framework based on java. A simulator is also provided to simulate ubiquitous programs by changing contexts.

Key Words : Context awareness computing, Context adaptation system, Adaptation rules, JCAF

1. 서 론

사람은 오감을 통하여 주변의 여러 가지 상황들을 인지하고 그에 대한 다양한 정보를 모아 그것을 바탕으로 해서 살아간다. 이와 마찬가지로 소프트웨어도 단순히 입력된 정보만을 이용하여 수동적으로 서비스하는 것에서 벗어나, 좀더 적극적으로 주변의 상황 정보들을 모으고 분석하여 상황에 따라 보다 적절한 서비스를 해보자는 것이 상황 인식 컴퓨팅의 기본 개념이다[5,6,7]. 상황 인식 컴퓨팅은 변화하는 상황을 인지하고 분석하여 그에 따라 적절히 반응함으로써, 변화하는 상황에 자동적으로 적응하여 한 차원 높은 서비스를 제공할 수 있다[5,6,7].

본 논문에서는 이처럼 변화하는 상황에 자동적으로 적응하는 프로그램을 보다 쉽게 개발할 수 있도록 도와줄 수 있는 상황 적응 시스템을 개발하였다. 본 시스템은 현실 상황이 변화할 때 프로그램이 상황 적응 규칙에 따라 자동적으로 적절히 반응하도록 함으로써 프로그램의 상황 적응 능력을 향상시킨다. 본 시스템을 이용하면 프로그래머는 상황에 따른 반응 혹은 적응 정책을 정책 기술 언어를 이용하여 간단히 기술하면 된다. 실제 프로그램 실행 과정에서 상황이 변화하면 시스템은 변한 상황에 대한 해당 적응 규칙을 적용함으로써 자동적으로 적절히 반응하도록 한다.

본 시스템은 변화하는 상황에 적응하기 위한 규칙을 기술하기 위해서 [1]에서 정의된 정책 기술 언어(Policy Description Language)를 사용한다. 이를 이용하여 작성된 상황 적응 정책은 정책 파일 리더기에 의해서 그 내용이 분석되며, 분석된 내용은 상황 적응 엔진에게 전달된다. 상황 적응 엔진은 분석된 상황 적응 정책의 내용에 따라 변화하는 현실 상황에 따라 그에 해당하는 반응을 자동적으로 하게 된다. 개발자는 상황 적응 정책만을 간단히 기술함으로써 자동적으로 자신의 유비쿼터스 프로그램을 변화하는 상황에 적응하도록 할 수 있다. 또한 본 시스템은 개발된 유비쿼터스 프로그램을 가상 상황에 따라 시뮬레이션 할 수 있는 시뮬레이터를 제공한다.

※ 본 연구는 숙명여자대학교 2007년도 교내연구비 지원에 의해 수행되었음.
† 정 회 원 : LG전자 연구원(본 연구는 숙명여자대학교 재학 중에 수행되었음)
†† 정 회 원 : 숙명여자대학교 컴퓨터과학과 교수(교신저자)
논문접수 : 2007년 8월 21일, 심사완료 : 2007년 10월 8일

본 시스템은 [2]에서 제안된 자바 기반 상황 인식 프로그래밍 프레임워크인 JCAF(Java Context Awareness Framework)를 이용해서 구현하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 상황 적응과 관련된 다른 연구들과 JCAF에 대해서 기술한다. 3장에서는 상황 적응 정책을 기술하기 위한 정책 기술 언어에 대해서 살펴볼 것이며, 4장에서는 정책 파일 리더기와 상황 적응 엔진의 구현에 대해서 기술한다. 5장에서는 본 시스템을 이용하여 개발된 프로그램을 가상에서 시뮬레이션 해 본 예제들을 살펴볼 것이며, 마지막으로 6장에서는 이 논문에 대한 결론을 맺는다.

2. 관련 연구

상황 적응을 지원하는 상황 인식 컴퓨팅 환경에 대한 몇몇 연구들이 진행되었다. 이 연구들과 본 연구의 차이점을 살펴보고, 본 시스템을 구현하는데 사용된 JCAF에 대해서 알아본다.

A. Ranganathan과 Roy H. Campbell이 Gaia 위에서 개발한 상황 인식 시스템[4]은 상황에 따라 나타나야 하는 반응을 1차 논리 구조(first-order logic)에 기반 하여 작성하고 처리한다. 그러나 개체 하나하나에 대하여 나타내고자 하는 상황과 반응을 모두 일일이 기술해 주어야 한다는 단점이 있다. 반면, 본 시스템에서는 같은 종류의 개체들을 집합적으로 표현할 수 있어, 원하는 상황 적응 규칙을 간단명료하게 작성할 수 있다.

Bellavista와 Montanari가 개발한 상황 인식 자원 관리 미들웨어 CARMAN[3]은 메타데이터의 내용을 기반으로 하여 무선인터넷 환경에서 변화된 환경을 재구성한다. 그러나 모든 유비쿼터스 환경에 적용되는 것이 아닌 무선인터넷 환경만으로 한정된다는 단점이 있으며, 메타데이터를 이용하여 원하는 것을 작성해야 한다는 차이점이 있다. 반면, 본 시스템에서는 모든 유비쿼터스 환경에 대해서 적용이 가능하며, 상황 적응 정책을 정의하는 것은 메타데이터를 작성하는 것보다 훨씬 간단하다.

Bardram이 개발한 Java 기반 상황 인식 프로그래밍 프레임워크인 JCAF는 상황 정보를 효과적으로 모아서 관리할 수 있도록 하는 기반 구조와 프로그래밍 인터페이스를 제공한다[2]. 그러나 단순히 상황 인식을 위한 프로그래밍 환경만을 제공할 뿐이지, 상황에 쉽게 적응할 수 있도록 하는 기능을 따로 제공하는 것은 아니다. 반면, 본 논문의 상황 적응 시스템은 상황 적응 정책만을 간단히 기술함으로써 상황에 쉽게 적응할 수 있도록 하는 기능을 제공한다. 본 시스템은 이 JCAF를 이용하여 구현되었으며, 상황 적응 정책을 기술하기만 하면 유비쿼터스 프로그램을 쉽게 상황에 적응시켜 준다.

JCAF(Java Context Awareness Framework)는 상황 인식 프로그램 개발을 위한 하부 구조와 API를 제공하는 자바 기반의 상황 인식 프로그래밍 프레임워크이다. JCAF는 현실의 상황을 가상으로 표현하기 위한 Entity, Context, Relationship, ContextItem 등의 라이브러리를 제공하며, 이렇게 표현된 상황 정보를 유지하고 관리하기 위한 ContextService와 상황 변화를 감지하기 위한 EntityListener 등의 라이브러리를 제공한다. 본 시스템은 JCAF가 제공하는 이 라이브러리들을 이용하여 현실의 상황을 가상으로 표현하고 관리한다.

3. 정책 기술 언어

유비쿼터스 프로그램을 변화하는 상황에 어떻게 적응시킬지 정의하기 위하여 본 시스템은 [1]에서 제안된 정책 기술 언어를 사용한다. 개발자는 이 정책 기술 언어에 따라 상황 적응 정책을 정의함으로써 자신의 유비쿼터스 프로그램을 변화하는 상황에 간단히 적응시킬 수 있다. 먼저 상황을 표현하는 방법에 대해 살펴보고, 상황 적응 정책을 어떻게 정의하는 지에 대하여 설명한다.

3.1. 상황 표현

상황 적응 정책에서 상황에 따른 적절한 반응을 정의하려면 먼저 상황을 표현할 수 있어야 한다. 상황은 엔티티들간의 관계에 의해서 형성된다. 엔티티는 현실에서의 실제적인 개체로서 PDA나 사람처럼 움직이는 것과, 빌딩이나 방과 같이 움직이지 않는 것이 있다. 이러한 엔티티들은 서로 관계를 맺어가며 여러 가지 상황들을 형성한다. 예를 들어, '방에 사람이 들어가다'라는 관계가 형성될 수 있고, '사람은 PDA를 가지다'라는 관계가 형성될 수 있다. 이러한 엔티티와 그 관계를 표현하기 위한 정책 기술 언어는 (그림 1)과 같은 문법 구조를 갖는다.

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

(그림 1) 상황 표현을 위한 문법 구조

Entity-Expression은 한 개의 엔티티나 엔티티의 집합을 표현한다. id1:id2는 한 개의 엔티티를 표현하기 위한 식으로, id1이라는 이름을 갖는 엔티티 클래스의 인스턴스 id2를 의미한다. 예를 들어 아래와 같이 나타내면 여러 방들 중에서 정확히 Tom의 방만을 표현하게 된다. 상황 적응 정책에서 이 식이 쓰였을 때 이것은 정확히 Tom의 방을 가리키게 된다.

Room:TomRoom

$id는 같은 종류의 엔티티들을 표현하기 위한 변수 식으로, id라는 이름을 갖는 엔티티 클래스의 어떤 인스턴스를 나타낸다. 예를 들어 아래와 같은 식은 방이라는 타입의 엔

티티를 모두 표현한다. 상황 적응 정책 안에서 이 식이 쓰였을 때 이것은 그 어떤 방이라도 가리킬 수 있게 된다.

$Room

$id_n은 id라는 이름을 갖는 엔티티 클래스의 인스턴스 중 어떤 하나와 그것과는 또 다른 하나를 표현하고자 할 때 쓰인다. 예를 들어 상황 적응 정책 안에서 아래와 같은 표현이 사용된다면 이 두 가지는 서로 다른 방을 의미하게 된다.

$Room_1, $Room_2

p1/p2는 공간을 표현하기 위한 식으로, 엔티티들의 종속적인 포함 관계를 나타내게 된다. 일반적으로 공간은 항상 종속적인 포함관계를 가지므로 그 구조가 '/'를 이용하여 표현될 수 있으며, 최종 공간에 위치한 일반 엔티티까지도 함께 표현할 수 있다. 예를 들어 UbiSoft라는 건물의 1층에 있는 방은 아래와 같은 식으로 나타낼 수 있다.

Building:UbiSoft/Floor:fl1/$Room

또한 엔티티의 종속적인 포함 관계를 좀 더 효율적으로 표현하기 위해 .../p 와 *를 제공한다. '...'은 생략기호로서 이것을 이용하여 공간적인 종속관계를 생략하여 표현할 수 있다. 예를 들어 다음 식은 UbiSoft라는 건물에 있는 어떤 방을 의미한다.

Building:UbiSoft/.../$Room

또한 '*'는 어떤 엔티티라도 모두 나타내기 위한 것으로, 모든 종류의 엔티티 클래스를 의미한다. 예를 들어 다음의 식은 UbiSoft라는 건물에 있는 모든 엔티티를 의미한다. 상황 적응 정책에서 이 식이 쓰였을 때 이것은 UbiSoft라는 건물에 있는 어떤 엔티티라도 가리킬 수 있게 된다.

Building:UbiSoft/.../*

Relation-Expression은 엔티티들간의 관계를 표현하기 위한 식으로 id1(p1,id2,p2) 구조를 갖는다. id1은 관계의 종류를 나타내고, p1은 주체가 되는 엔티티를 나타내며 id2는 관계를 나타내고 p2는 대상이 되는 엔티티를 나타낸다. 예를 들어 '어떤 사람이 Tom의 방에 있다'라는 관계는 아래와 같이 표현된다.

Location($Person, IsIn, Room:TomRoom)

또한 이 식에는 NOT을 의미하는 '~'기호와 AND를 의미하는 '^'기호를 사용할 수 있다. 예를 들어 다음의 식은 '어떤 사람이 Tom의 방에 있는데, 그 사람이 Tom의 방의 주인이 아닌 상황'을 의미한다.

Location($Person, IsIn, Room:TomRoom) ^
~Ownership($Person, Owns, Room:TomRoom)

이 식은 참·거짓으로 해석 될 수 있으며, 상황 적응 정책에서는 이 식이 조건을 표현하는 데 쓰인다.

3.2. 상황 적응 정책

상황 적응 정책은 변화하는 상황에 어떻게 적응해야 할 것인가에 대하여 정의한 것이다. 특정한 상황들에 대하여 그에 맞는 적절한 반응들을 기술함으로써 상황 적응 정책을 정의하게 된다. 이를 위한 문법 구조는 (그림 2)와 같다.

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

(그림 2) 상황 적응 정책의 문법 구조

Adaptation-Rule은 상황에 따라 어떻게 반응해야 할지를 기술하기 위한 식으로, r => a 라는 구조를 가진다. 이 식은 상황 r에서 반응 a를 보이라는 것으로, r이라는 조건이 참이 되면 a라는 행동을 실행하라는 것을 의미한다. r은 앞서 설명했던 Relation-Expression을 이용하여 표현되며, a는 Action을 이용하여 표현된다.

Action의 종류에는 메소드 호출과 새로운 관계의 생성이 있으며, ';' 기호를 이용하여 여러 개의 행동을 하게 할 수도 있다. 메소드 호출은 p1.id(p2)의 구조로 나타낸다. id는 메소드의 이름을 의미하며 p1은 그 메소드를 가지고 있는 엔티티를 의미하고, p2는 메소드의 매개변수로 넘겨지는 또 다른 엔티티를 의미한다. 예를 들어, PDA에 어떤 사람의 정보를 자동으로 보여주고자 하면 아래와 같은 메소드 호출을 사용하면 된다.

$Pda.showInfo($Person)

새로운 관계의 생성은 id1(p1,id2,p2)의 구조를 갖는데, 이는 Relation-Expression에서와 같은 형태를 가진다. 예를 들어, 가게에 어떤 사람이 들어왔을 때 그 사람을 가게의 손님으로 지정하고 싶다면 아래와 같이 새로운 관계를 형성해 주면 된다.

Connection($Person, IsGuestOf, $Shop)

이렇게 원하는 상황을 조건으로 표현하고 그때 나타나기 원하는 반응을 행동으로 기술함으로써, 상황 적응 정책을 정의할 수 있다. 예를 들어, 'UbiSoft라는 빌딩의 사무실에 사람이 들어오면 관리자의 PC에 그 사람의 정보를 보여준다.' 라는 상황 적응 정책은 아래와 같이 작성될 수 있다. 이것은 UbiSoft라는 빌딩의 사무실에 어떤 사람이 들어왔다는 상황 변화가 일어났을 때 자동으로 실행되어 관리자의 PC에 그 사람의 정보를 보여주게 된다.

Location($Person, IsIn, Building:UbiSoft/.../$Room) ^
Ownership($Admin, Owns, $PC)
=> $PC.showInfo($Person)

또 다른 예로, '비디오 가게에 들어온 사람이 어른이라면 그를 그 가게의 손님으로 설정한다.' 라는 상황 적응 정책은 아래와 같이 작성될 수 있다. 이것은 눈에 보이는 어떤 행동을 해주지는 않지만 어른이 들어왔을 때만 그를 그 가게의 손님으로 맞아들이는 새로운 상황 변화를 만들어 낸다.

```
Location($Person, IsIn, $VideoShop) ^
Status($Person, Is, $Adult)
=> Connection($Person, IsGuestOf, $VideoShop)
```

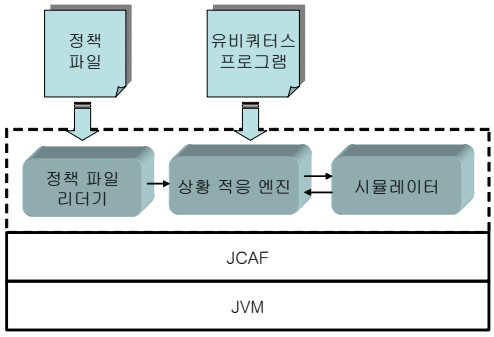
이처럼 하나의 규칙은 새로운 관계를 생성해 낼 수 있기 때문에, 이것이 또 다시 상황을 변화시켜 또 다른 규칙이 자동으로 실행될 수 있다. 예를 들어, '비디오 가게에 손님이 들어온 경우 환영 메시지를 내보낸다.' 라는 상황 적응 정책이 아래와 같이 작성됐다면, 이는 위의 상황 적응 정책이 실행되어 현재 들어온 사람이 손님으로 설정이 되었을 경우에만 자동으로 실행되게 된다.

```
Location($Person, IsIn, $VideoShop) ^
Connection($Person, IsGuestOf, $VideoShop)
=> $VideoShop.speakMessage($Person)
```

4. 시스템 설계 및 구현

유비쿼터스 프로그램을 변화하는 상황에 적응시키기 위한 상황 적응 시스템은 (그림 6)과 같은 구조를 갖는다.

상황 적응 시스템은 상황 적응 정책이 기술되어 있는 정책 파일을 읽고 분석하기 위한 정책 파일 리더기와, 그 분석 내용에 따라 유비쿼터스 프로그램을 상황에 적응시키기 위한 상황 적응 엔진으로 구성된다. 또한 가상으로 상황 변화에 적응시켜볼 수 있는 시뮬레이터를 제공한다. 본 시스템은 자바 기반의 상황 인식 프로그래밍 프레임워크인 JCAF[2] 위에서 구현되었다.



(그림 3) 상황 적응 시스템

4.1. 정책 파일 리더기

정책 파일 리더기는 정책 파일의 상황 적응 정책을 클래스의 형태로 분석한다. 상황 적응 정책들을 이렇게 클래스 형태로 분석하고 저장함으로써 상황 적응 과정을 좀 더 체계적으로 관리할 수 있게 된다. 조건을 검사하고 행동을 실행하는 복잡한 상황 적응 과정은 그에 필요한 작업들을 각각의 적합한 클래스들로 분산시킴으로써 더욱 체계적이고 명확해 진다.

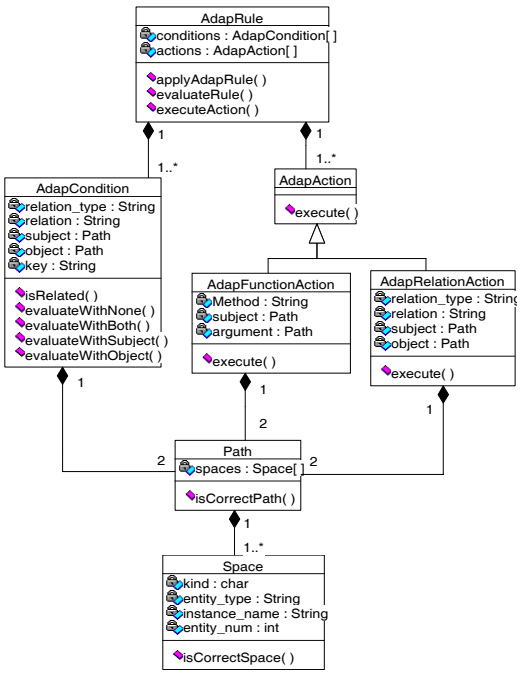
정책 파일 리더기는 정책 파일을 분석하기 위하여 (그림 4)에 있는 것과 같은 메소드를 제공한다. 매개변수로 정책 파일을 넘기게 되면 분석된 상황 적응 정책을 얻을 수 있다. 상황 적응 엔진은 이 메소드를 호출하여 정책 파일을 분석하고 그 결과를 얻는다.

```
Rule[] analyze(String policy_file);
```

(그림 4) 정책 기술 언어 분석 메소드

(그림 5)는 정책 파일을 분석한 Rule 클래스의 구조를 나타내는 클래스 다이어그램이다. AdapRule은 상황 적응 정책을 나타내는 클래스이다. 이 클래스는 조건을 의미하는 AdapCondition과 그에 따른 행동을 의미하는 AdapAction을 가진다. AdapAction에는 메소드 호출을 의미하는 AdapFunctionAction과 새로운 관계 형성을 의미하는 AdapRelationAction이 올 수 있다. 엔티티들의 공간적인 포함구조는 Path라는 클래스로 표현되며, 하나하나의 공간은 Space로 표현된다.

이러한 클래스들은 각자 자신이 필요한 정보들을 저장하고 관리하며, 상황 적응 과정 시 각각 필요한 행동들을 직접 수행함으로써 그 과정을 더욱 명확하고 체계적으로 만들어 준다.



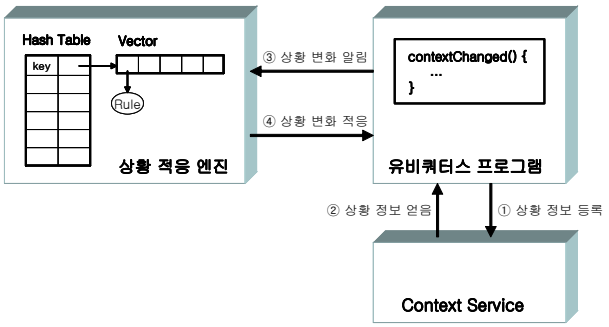
(그림 5) Rule 클래스의 구조

4.2. 상황 적응 엔진

상황 적응 엔진은 정책 파일 리더기가 분석해 준 상황 적응 정책의 내용을 기반으로 하여 유비쿼터스 프로그램을 변화하는 상황에 적응시킨다. 상황 적응 엔진은 자바 기반의 상황 인식 프로그래밍 프레임워크인 JCAF를 이용하여 유비쿼터스 프로그램의 상황 변화를 인식하고 거기에 반응한다.

4.2.1. 유비쿼터스 프로그램과의 연동

(그림 6)은 상황 적응 엔진이 JCAF에서 제공되는 상황 인식 구조를 이용하여 유비쿼터스 프로그램과 연동되는 모습을 보여주고 있다. JCAF에서 제공되는 ContextService는 유비쿼터스 프로그램의 상황 정보를 관리해 준다.



(그림 6) 유비쿼터스 프로그램과의 연동 모습

상황 적응 엔진과 유비쿼터스 프로그램의 연동 과정은 다음과 같다.

1) 유비쿼터스 프로그램은 JCAF의 ContextService에 상황 정보를 등록한다.
2) ContextService는 상황에 변화가 발생했을 때 유비쿼터스 프로그램에게 알린다.
3) 유비쿼터스 프로그램은 다시 그 변화를 상황 적응 엔진에게 알린다.
4) 상황 적응 엔진은 상황 적응 정책의 내용에 따라 유비쿼터스 프로그램을 그 변화에 적응시킨다.

상황 정보를 관리해 주는 ContextService는 상황이 변화하였을 때 유비쿼터스 프로그램의 contextChanged() 메소드를 호출함으로써 이를 알려준다. 유비쿼터스 프로그램은 contextChanged() 메소드에서 상황 적응 엔진의 adaptContext() 메소드를 호출함으로써 상황 적응 엔진에게 상황 변화를 알리고 적응을 요청하게 된다. (그림 7)은 이 메소드의 형태를 보여준다. 변화된 상황에 대한 정보가 이 메소드의 인자로서 넘겨지게 된다.

```
void adaptContext(Entity e, Relationship r, ContextItem i)
```

(그림 7) 상황 적응 메소드

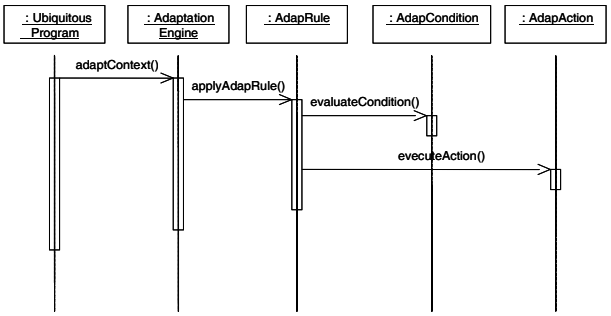
4.2.2. 상황 적응 과정

상황 적응 엔진은 다음과 같은 과정을 거쳐 유비쿼터스 프로그램을 변화하는 상황에 적응시킨다.

1) 정책 파일 리더기로부터 상황 적응 정책이 분석된 Rule 클래스 정보를 받는다.
2) Rule 클래스들을 해시테이블에 저장한다.
3) 유비쿼터스 프로그램이 상황 변화를 알려올 때까지 대기한다.
4) 유비쿼터스 프로그램이 상황 변화를 알려오면 해시테이블에서 변화와 관련된 Rule 클래스들을 찾는다.
5) 검색된 Rule들에 대해서 조건이 참인지 검사한다.
6) 조건이 참의 값을 갖는다면 그 Rule의 행동을 실행시킨다.
7) 다시 유비쿼터스 프로그램이 상황 변화를 알려오기를 기다린다.

이와 같이 상황 적응 엔진은 상황에 변화가 일어났을 때, 그와 관련된 상황 적응 정책이 있는지 찾아보고 그것을 적용시켜 줌으로써 유비쿼터스 프로그램을 상황 변화에 적응시킨다.

상황 적응 엔진은 상황 적응 과정에 필요한 작업들을 관련된 각각의 클래스들에 분산시킴으로써 복잡한 작업을 단순하게 한다. (그림 8)은 상황에 적응하는 과정에서 나타나는 각 클래스들 사이의 관계를 개략적으로 보인 것이다. 먼저 유비쿼터스 프로그램이 상황 적응 엔진에게 상황 변화에의 적응을 요청하고, 상황 적응 엔진은 변화와 관련된 Rule을 찾아 적용시켜보게 된다. 해당 Rule은 조건이 참의 값을 갖는지 검사해보고, 참의 값을 갖는 경우에 그 행동을 실행시킨다.



(그림 8) 상황 적응 과정

4.2.3. 해시테이블의 관리

상황 적응 엔진은 Rule 클래스들을 효과적으로 관리하고 변화된 상황과 관련된 Rule을 빠르게 찾기 위하여 해시테이블을 사용한다. 변화된 상황과 관련된 Rule을 빠르게 찾아야 하므로, Rule의 조건에서 상황과 관련된 부분을 키로 만들어 사용한다. Rule의 조건은 복합적인 여러 가지 상황에 의해 표현될 수 있기 때문에, 하나의 Rule은 여러 개의 키

에 의해서 가리켜질 수 있다. 또한 하나의 상황이 여러 Rule들에 의해서 조건을 표현하는데 쓰일 수 있으므로, 하나의 키에 대해서 거기에 관련된 여러 개의 Rule들을 Vector로 묶어서 저장한다. 이러한 해시테이블의 형태가 (그림 6)의 상황 적응 엔진에 나타나 있다.

상황은 엔티티들간의 관계로서 표현이 된다. 즉, 주체 엔티티와 대상 엔티티 사이에서의 관계로 상황을 표현하게 된다. 따라서 해시테이블의 키는 '주체 엔티티, 대상 엔티티, 관계' 이 세 가지 주요 요소를 사용하여 만든다. 처음에 상황 적응 정책들을 해시테이블에 저장할 때는 Rule의 조건들에서 위의 주요 요소들을 뽑아 키로 만들고, 상황에 변화가 일어나 해시테이블에서 Rule을 검색할 때는 변화를 일으킨 상황 정보에서 위의 주요 요소들을 뽑아 키로 만든다.

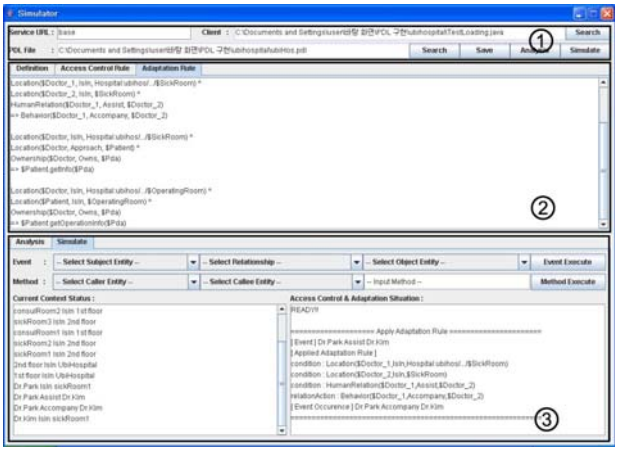
예를 들어, '사람이 방에 있으면'이라는 조건을 가지는 Rule을 해시테이블에 저장한다고 하면, Rule의 조건인 Location($Person, IsIn, $Room)에서 핵심 요소인 Person, IsIn, Room만을 추출하여 Person:IsIn:Room 이라고 키를 만들어 저장하게 된다. 그러다 유비쿼터스 프로그램이 Tom이 BobRoom에 들어왔다는 상황 변화를 알리게 되면, adapContext() 메소드의 실 매개변수로서 넘어온 Tom, IsIn, BobRoom 이라는 상황 변화 정보를 이용하여 키를 만들게 된다. Tom은 Person의 객체이고 BobRoom은 Room의 객체이므로, 상황 적응 엔진은 Person:IsIn:Room이라고 키를 만들어 해시테이블을 검색하게 된다. 그러면 앞서 저장한 Rule이 찾아질 것이고 이 Rule에 대해서 조건을 검사하고 행동을 실행시켜주게 된다.

4.3. 시뮬레이터

시뮬레이터는 개발자로 하여금 다양한 시나리오를 가지고 가상으로 유비쿼터스 프로그램을 시뮬레이션해 볼 수 있도록 도와준다. 시뮬레이터를 이용하여 프로그래머는 가상으로 유비쿼터스 프로그램의 상황을 변화시키고, 상황 적응 정책에 따라 상황에 따라 적응하는 모습을 볼 수 있다[8].

(그림 9)는 시뮬레이터를 실행한 모습이다. 시뮬레이터는 크게 세 부분으로 구성된다. 시뮬레이션 하는데 필요한 기본적인 정보를 설정하는 ① 부분, 상황 적응 정책의 내용을 보여주는 ② 부분, 상황을 변화시키고 그 내용을 확인해 볼 수 있는 ③ 부분으로 구성된다.

기본적인 정보를 설정하는 ① 부분은 사용할 ContextService의 URL을 입력하는 부분과 시뮬레이션 해 보고 싶은 유비쿼터스 프로그램과 정책 파일을 선택하는 부분으로 구성되어 있으며, 시뮬레이션을 시작하고 수정한 정책 파일을 저장할 수 있도록 해주는 버튼들로 구성되어 있다. 상황 적응 정책의 내용을 보여주는 ② 부분은 그 내용을 시뮬레이터에서 바로 수정해 볼 수 있도록 설계되었으며, ③ 부분은 상황을 변화시킬 수 있도록 이벤트를 발생시키는 부분과 그 변화된 내용을 확인하는 부분으로 구성되어 있다. Current Context Status 화면에서는 변화된 상황 정보가 보여 지며, Adaptation Situation 화면에서는 상황 적응 정책에 따라 상황에 적응된 모습이 보여 진다.



(그림 9) 시뮬레이터

개발자는 자신의 유비쿼터스 프로그램을 변화하는 상황에 가상으로 적응시켜 보기 위해 다음과 같은 순서로 시뮬레이터를 이용할 수 있다.

1) Service URL과 유비쿼터스 프로그램, 정책 파일을 선택한다.
2) 정책 파일을 수정할 필요성이 있다면 수정하고 저장한다.
3) simulate 버튼을 눌러 시뮬레이션을 시작한다.
4) Current Context Status 화면에서 현재의 상황 정보를 확인하고, 상황을 변화시키고 싶은 이벤트를 발생시킨다. 원하는 주체 엔티티와 릴레이션, 대상 엔티티를 선택하고 Event Execute 버튼을 누르면 해당하는 이벤트가 발생한다.
5) Current Context Status 화면에서 변화된 상황 정보를 확인하고, Adaptation Situation 화면에서 변화된 상황에 적응하여 실행된 상황 적응 정책이 있는 확인한다.

이와 같은 방법으로 다양한 시나리오를 통해 가상에서 유비쿼터스 프로그램을 시뮬레이션 해 봄으로써 자신의 프로그램을 더욱 견고하게 만들 수 있다.

5. 실 험

5.1. 유비쿼터스 병원

병원은 순간순간 상황이 매우 급변하며 상황에 따라 많은 일들이 일어나는 곳으로 이를 위한 상황 인식 및 적응 프로그램을 구현하는 것은 매우 유용한 일이다.

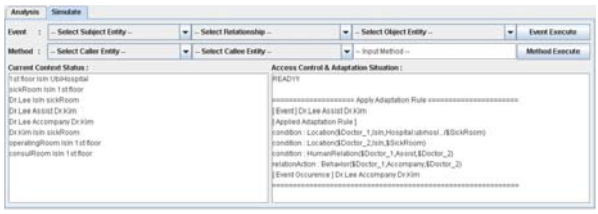
(그림 10)은 유비쿼터스 병원을 위한 상황 적응 정책의 예이다. 이 예에서는 네 가지 상황 적응 정책을 정의하고 있다. 첫 번째는 의사가 있는 진료실에 환자가 들어오면 의사의 PDA에 자동으로 환자의 정보를 보여주도록 하는 정책이다. 두 번째는 의사가 입원실에 들어가서 환자에게 다가가면 해당 환자의 정보를 자동으로 의사의 PDA에 보여주도

록 하는 정책이다. 세 번째는 보조 의사가 주치의와 함께 입원실에 들어오면 동행 관계를 생성해 주는 정책이다. 마지막은 의사가 수술실에 들어오면 환자의 수술정보를 자동으로 PDA에 보여주는 정책이다.

```
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 10) 유비쿼터스 병원의 정책 파일

(그림 11)은 이 예제를 시뮬레이터를 이용하여 가상에서 실행시켜본 것이다. 보조 의사가 주치의와 함께 입원실에 들어왔을 때 동행관계가 형성되는 모습이 보여 지고 있다.



(그림 11) 시뮬레이터 실행 화면

5.2. 유비쿼터스 빌딩

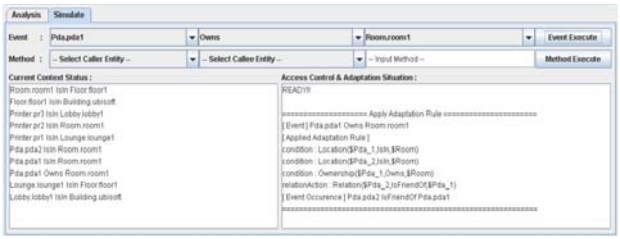
많은 사람들이 오고 가는 빌딩 같은 경우에도 자동적으로 상황에 적응하는 상황 인식 프로그램을 이용하면 효과적으로 빌딩의 상황을 관리하고 편리하게 서비스를 제공할 수 있다.

(그림 12)는 유비쿼터스 빌딩에 대한 정책 파일의 예이다. 이 파일에는 5 가지의 상황 적응 정책이 정의되어 있다. 첫 번째 정책은 어떤 PDA가 방에 들어오면, 방 주인의 PDA는 들어온 PDA를 친구로 등록한다. 두 번째 정책은 방 주인의 PDA가 방에 들어오면, PDA에 그 방의 프린터를 기본 프린터로 설정한다. 세 번째 정책은 방 주인 친구의 PDA가 방에 들어오면, 그 친구의 PDA에 그 방의 프린터를 기본 프린터로 설정한다. 네 번째와 다섯 번째 정책은 로비와 라운지에 어떤 PDA가 들어오면, 그 PDA에 그곳의 프린터를 기본 프린터로 설정한다.

(그림 13)은 이 예제를 시뮬레이터를 이용하여 가상에서 실행시켜본 것이다. 방에 손님이 들어왔을 때, 방 주인의 PDA가 손님의 PDA를 손님으로 설정하는 모습이 보여 지고 있다.

```
Location($Pda_1, IsIn, $Room) ^ Location($Pda_2, IsIn, $Room)^
Ownership($Pda_1, Owns, $Room)
    => Relation($Pda_2, IsFriendOf, $Pda_1)
Location($Pda, IsIn, Building:ubisoft/.../$Room) ^
Ownership($Pda, Owns, $Room)
    => $Pda.registerPrinter($Room/$Printer)
Location($Pda_1, IsIn, Building:ubisoft/.../$Room) ^
Ownership($Pda_2, Owns, $Room) ^
Relation($Pda_1, IsFriendOf, $Pda_2)
    => $Pda_1.registerPrinter($Room/$Printer)
Location($Pda, IsIn, Bilding:ubisoft/$Lobby)
    => $Pda.registerPrinter($Lobby/$Printer)
Location($Pda, IsIn, Building:ubisoft/$Lounge)
    => $Pda.registerPrinter($Lounge/$Printer)
```

(그림 12) 유비쿼터스 빌딩의 정책 파일



(그림 13) 시뮬레이터 실행 화면

5.3. 유비쿼터스 대학

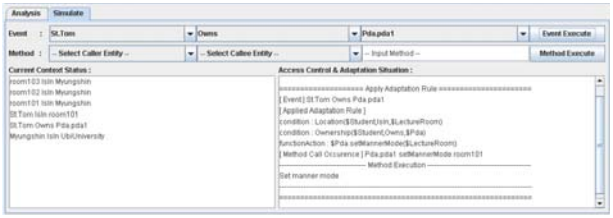
대학에서는 많은 강의가 개설이 되고, 많은 학생들과 강사들이 그 강의에 참가를 한다. 또한 많은 행사가 열리며, 동아리 등의 수업 외 활동도 활발하다. 이렇게 상황이 쉼 없이 변하는 대학 같은 경우에도 역시 자동적으로 상황에 적응하는 상황 인식 프로그램을 이용하면 효과적으로 대학을 관리하고 편리하게 서비스를 제공할 수 있다.

(그림 14)는 유비쿼터스 대학에 대한 정책 파일의 예이다. 이 파일에는 세 가지의 상황 적응 정책이 정의되어 있다. 첫 번째는 어떤 학생이 강의실에 들어왔는데 그 강의실에서 강의가 열리고 있으면, 그 학생이 그 강의에 참석한 것으로 한다는 정책이다. 두 번째는 학생이 강의에 참석하고 있고, 그 학생이 PDA를 가지고 있다면, 그 PDA에 강의 자료를 자동으로 보여주라는 정책이다. 마지막은 PDA를 가진 학생이 강의실에 들어가면 자동으로 PDA를 매너모드로 바꾸도록 하는 정책이다.

(그림 15)는 이 예제를 시뮬레이터를 이용하여 가상에서 실행시켜본 것이다. PDA를 가진 학생이 강의실에 들어갔을 때 자동으로 PDA가 매너모드로 바뀌는 모습을 보여주고 있다.

```
Location($Student, IsIn, $LectureRoom) ^
Class($Lecture, Open, $LectureRoom)
    => Class($Student, Attend, $Lecture)
Class($Student, Attend, $Lecture) ^
Ownership($Student, Owns, $Pda)
    => $Pda.registerLecture($Lecture)
Location($Student, IsIn, $LectureRoom) ^
Ownership($Student, Owns, $Pda)
    => $Pda.setMannerMode($LectureRoom)
```

(그림 14) 유비쿼터스 대학의 정책 파일



(그림 15) 시뮬레이터 실행 화면

6. 결 론

본 논문에서는 프로그램 개발자가 정의한 정책에 따라 유비쿼터스 프로그램을 변화하는 상황에 적응시키는 상황 적응 시스템을 개발하였다. 본 시스템은 정책 기술 언어[1]에 따라 작성된 상황 적응 정책을 읽고 분석하기 위한 정책 파일 리더기를 제공하며, 상황 적응 정책의 내용에 따라 유비쿼터스 프로그램을 변화하는 상황에 적응시키기 위한 상황 적응 엔진을 제공한다. 또한 본 시스템을 이용하여 작성된 유비쿼터스 프로그램을 가상 상황에서 시뮬레이트 하기 위한 시뮬레이터를 제공한다.

프로그램 개발자는 본 시스템을 이용하여 변화하는 상황에 자동으로 적응하는 유비쿼터스 프로그램을 보다 쉽게 개발할 수 있으며, 이렇게 개발된 프로그램은 변화하는 상황에 적응하여 필요한 것을 먼저 서비스함으로써 한 차원 높은 서비스를 제공할 수 있다.

참 고 문 헌

[1] J. Ahn, B.-M. Chang, and K.-G. Doh, A Policy Description Language for Context-based Access Control and Adaptation in Ubiquitous Environment, *TRUST 2006*, August, 2006.

[2] J. E. Bardram, The Java Context Awareness Framework-A Service Infrastructure and Programming Framework for Context-Aware Applications, *Pervasive 2005*, Munich, Germany, May, 2005.

[3] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, Cesare Stefanelli, Context-Aware Middleware for Resource Management in the Wireless Internet, *IEEE Trans. Software Eng.* 29(12): 1086-1099 (2003).

[4] A. Ranganathan, Roy H. Campbell, An infrastructure for context-awareness based on first order logic, *Personal and Ubiquitous Computing* 7(6): 353-364 (2003).

[5] T. Moran and P. Dourish. Introduction to this speical issue on context-aware computing. *Human-Computer Interaction*, 16:87-95, 2001.

[6] H. Lei, D. M. Sow, I. John S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):45-55, 2002.

[7] M.Weiser. The computer for the 21st century. *Scientific American*, 265(3):66-75, September 1991.

[8] 이지연, 오민경, 창병모, 안준선, 도경구, 유비쿼터스 컴퓨팅을 위한 접근제어와 상황 적응 시스템, 한국정보과학회 가을학술발표논문집(B) 제33권 제2호, pp.590-594, 2006년 10월

오 민 경

e-mail : omk@sookmyung.ac.kr

2005년 숙명여자대학교 컴퓨터과학과 (이학사)

2007년 숙명여자대학교 컴퓨터과학과 (이학석사)

2007년~현재 LG 전자 연구원

관심분야 : 유비쿼터스 소프트웨어 및 시스템

창 병 모

e-mail : chang@sookmyung.ac.kr

1988년 서울대학교 컴퓨터공학과(공학사)

1990년 한국과학기술원 전산학과 (공학석사)

1994년 한국과학기술원 전산학과 (공학박사)

1995년~현재 숙명여자대학교 컴퓨터과학과 교수

관심분야 : 프로그래밍 언어 및 시스템, 유비쿼터스 소프트웨어