유비쿼터스 환경에서 Context 처리를 위한 패턴 수집 모델의 설계와 구현[1]

이대준○ 김성조
중앙대학교 컴퓨터공학과
vsjun@konan.cse.cau.ac.kr○, sjkim@cau.ac.kr

Design and Implementation of Pattern Collection Model for Handling the Context on a Ubiquitous Environment

Dae Jun Lee○ Sung Jo Kim
Dept. of Computer Science and Engineering, Chung-Ang University

요 약

유비쿼터스 환경에서 사용자의 편의성을 증대하기 위해 상황인지 기술이 필요하며 댁내의 다양한 기기의 정보를 수집하여 현재 상태를 파악하고 그에 맞는 서비스를 제공해야 한다. 하지만 다양한 종류의 디바이스, 센서, 서비스에 따라 생성하는 데이터의 형태와 의미가 다르기 때문에 이를 활용하는데 어려움이 있다. 본 논문은 다양한 기기와 환경에서 발생하는 데이터를 처리하여 사용자의 패턴을 수집하고 활용할 수 있는 연구 모델을 제안하고 구현한다. 구성요소는 실제 환경과 유사하게 Context를 생성할 수 있는 Emulator와 수집된 Context를 활용하여 패턴을 찾는 패턴 수집 서버와 수집된 데이터를 표현하는 시각화 도구로 구성된다. Emulator는 댁내에 존재할 수 있는 다양한 종류의 Context를 정의하고 서로간의 관계에 따라 Context를 생성하고 패턴 수집 서버는 Emulator에서 생성한 불완전한 Context를 통합하여 완전한 Context를 생성한다. 그리고 생성된 Context를 통해서 사용자의 서비스 이용패턴, Fault, Conflict를 발견했다.

1. 서 론

유비쿼터스 기술의 등장으로 인하여 가전기기나 정보기기, 네트워크 등의 시스템이 일상 생활에 내포되어 일상 가사 활동 외에도 엔터테인먼트, 오락 등 다양한 분야의 일들이 사용자의 행동에 대해 능동적으로 반응하고 더 나아가서는 사용자가 원하는 상황에 가장 적합한 서비스가 요구된다.

미래의 스마트 홈을 위한 기술은 사용자가 원하는 일을 보다 능동적으로 파악하여 작업을 수행할 수 있도록 매일 생활하는 실제 환경과 디지털 기술을 괴리감 없이 연결하기 위한 방향으로 진행되고 있다. 이러한 목적을 위해 현재 진행되고 있는 미래의 스마트 홈 환경을 위한 시스템에 대한 연구는 비전, 음성인식, 제스쳐 기반의 인터페이스를 통하여, 인간의 의도나 행동을 파악하여 모든 사항을 자동화하려고 시도되고 있다.[1][2] 하지만 다양한 댁내의 디바이스, 센서 시스템은 각각의 다른 표준을 사용하여, 서로 다른 의미의 데이터를 생산하며 이들 만으로는 댁내 전체의 상황을 파악하기 어렵다. 또한 편리성 측면에서 고장이나 Fault가 발생했을 때 이를 자동으로 감지하고 처리할 뿐 아니라 더 나아가 고장을 예측하고 발생하기 전에 처리하는 시스템이 요구된다.

그러나 현재 홈네트워크는 고장 발생에 능동적으로 대처하지 못한다. 고장을 처리하기 위해서는 고장의 발생을 미리 예측해서 처리하는 방식과 발생한 고장을 감지해서 처리하는 방식이 있다. 일반적으로 댁내 사용자는 일련의 작업을 순서대로 행동하거나 반복하는 패턴을 가질 확률이 높다. 예를 들어 매일 아침 일어나서 하는 행동이나 퇴근 후에 하는 행동이 유사할 수 있다. 또한 이런 사용자의 패턴이 고장의 원인이 될 수 있다. 사용자와 댁내에서 제공하는 서비스가 충돌을 하거나 사용자간 행위의 결과가 충돌하는 경우가 발생한다면, 이러한 사용자의 패턴이 고장의 원인이 된다고 할 수 있다.

이러한 서비스를 제공하고 연구하기 위하여 댁내에서 발생하는 Context를 수집, 조합하고 패턴을 찾는 모델을 제시한다.

본 논문의 구성은 다음과 같다. 제2장의 관련 연구에서는 댁내에서 발생한 Context처리에 관한 관련 연구를 알아보고 제3장에서는 논문에서 사용하는 Context, 패턴, 장애와 같은 기본 개념에 대해 알아본다. 제3장에서는 본 논문에서 제시하는 패턴 수집 모델에서의 패턴 수집 서버와 Emulator의 구조의 설계 및 구현에 대해 설명하며,

[1] 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 (홈네트워크연구센터) 지원사업의 연구결과로 수행되었음 (IITA-2006-C1090-0603-0035)

제4장에서는 구현한 구조를 테스트한 결과를 설명한다. 마지막으로 제5장에서는 결론과 향후 연구방향에 대하여 기술한다.

2. 관련 연구

댁내에서 발생하는 Context의 조합이나 결합으로 Context를 처리하는 연구가 이루어졌다. 초기의 연구[1]에서는 상호작용이 많은 애플리케이션에서 Context 정보를 다루기 위한 Context 결합에 대한 연구를 진행하였다. 연구의 목적은 다양한 Context를 수집을 하고 이를 애플리케이션에서 활용할 수 있는 데이터로 변형하는 것이었다. 이를 위해 다양한 종류의 센서에서 발생하는 Context를 처리하는 Context 툴킷을 제안했다. 이 연구에서는 거주 환경에서 인식해야 할 환경 정보의 결합에 대한 구조를 제시하였지만 주로 센서에서 발생하는 컨텍스트만을 고려하였으며 다양한 홈네트워크의 기기들을 고려하지 않았다. 이후 연구[2]에서는 다양한 Context를 조합하고 처리하는 구조를 제시하였다. 실제 기기에서 발생할 수 있는 데이터를 정리하며 이를 조합하여 활용할 수 있는 방안을 연구했고 Context의 결합과 관련된 기법과 관련 구조를 제시하였지만 사용자의 서비스 이용 패턴이나 댁내에서 발생할 수 있는 고장을 고려하지는 않았다.

이들 연구에서는 홈네트워크에서 Context를 활용하기 위한 구조에 관련한 연구가 진행되었다. 하지만 댁내에서 발생하는 다양한 종류의 데이터를 고려하지 않았거나, 사용자의 패턴이나 고장을 고려하지 않았다. 본 논문에서는 다양한 컨텍스트를 고려하고 댁내에서 발생하는 Context를 통해 사용자의 패턴, Fault를 찾을 수 있도록 한다.

3. 패턴 수집 모델의 기본 개념

3.1 Context

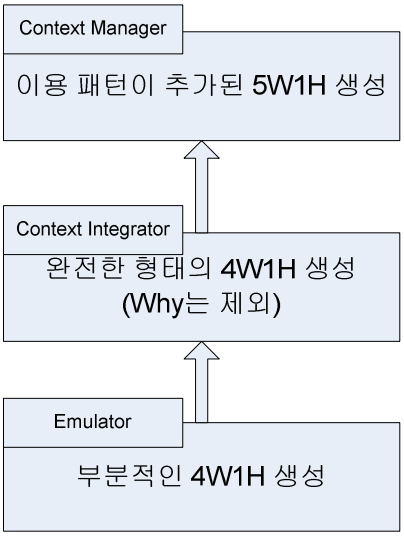
근래의 컴퓨팅 패러다임에서 Context는 주위 상황을 묘사하는 정보를 의미한다. 홈네트워크 환경에서 다양한 정보들이 존재하고 기술이 발달함에 따라 여러 상황을 표현할 수 있는 Context는 증가된다. 본 논문에서는 홈네트워크 환경에서 상황을 묘사하는 데이터를 Context라고 정의 하고 이를 통해 댁내 환경을 파악한다. Context는 댁내의 상황정보, 센서정보 등을 표시할 수 있는 방식을 필요로 하며, 이를 통해 댁내 상황을 파악에 필요한 정보를 제공한다.

댁내 환경을 표현하기 위해서는 여러 가지 Context가 사용될 수 있다. 하지만 표준 별, 기기 별로 자신의 상황을 표시하고 정의하는데 다양한 방식이 사용된다면 서로 다른 상황 데이터를 통합하는데 문제가 있다. 따라서 본 논문에서는 Context의 표현을 위해 육하원칙(5W1H)를 이용하는데, 그 표현 방식은 <표 1>과 같다.

각각의 디바이스, 센서, 서비스에서 생성하는 Context는 육하원칙으로 표현하기에는 데이터의 양이 충분하지 않다. 뿐만 아니라, 각자의 표준에 맞춰 생성하는 Context는 댁내의 전반적인 상황을 파악하기에는 충분하지 않다. 예를 들어, "사용자 <A>는 현재 거실에 있으며 TV의 볼륨을 10으로 조정하였다."라는 전체적인 상황을 홈네트워크 상에서 파악하기는 어렵다. 대신"<A>는 위치(a, b)에 있음", "TV의 볼륨이 10으로 조정됨"이라는 개별 디바이스, 센서에서 생성될 것이다. 따라서 개별 Context를 통합하여 전체 상황을 표현할 수 있는 Context를 만드는 행위가 필요하다. 그런데 예시로 든 상황에서 왜 사용자 <A>는 볼륨을 10으로 조정했는가에 대한 이유는 표현되지 않는다. 만일 "A는 거실에서 홈시어터를 이용해 영화를 볼 때 볼륨을 주로 10으로 설정한다"라는 패턴을 유지하고 있으면, Why에 사용자 행위의 이유로서 사용자의 패턴을 표현할 수 있다. 이러한 Context 생성과정을 정리하면 (그림 1)과 같다.

<표 1> Context 표현 방식

분류	내용
Where	Context를 표현하는데 있어서 가장 보편적인 정보로서 행위가 일어난 위치를 나타냄. 절대위치(좌표), 상대 위치(방, 거실) 등으로 표현
Who	사용자, 서비스와 같이 행위를 실시하는 주체에 대한 정보. 사용자 이름, 사용자 ID, 서비스이름, 서비스 ID등으로 표현
What	주체(사용자, 서비스)가 수행한 대상을 나타냄. 대상(디바이스, 센서, 네트워크) 이름, 대상 ID 등으로 표현
When	행위가 일어난 시간에 대한 정보. 절대시간, 상대시간, 기간 등으로 표현
How	대상이 수행한 행위(What)에 대한 정보. 장치의 상태변화, 사용자 이동 방향 등으로 표현
Why	상황이 발생했을 때 주체(Who)의 어떤 패턴에 의한 것인지에 대한 정보. "<A>라는 사용자는 퇴근 후 샤워를 한다"라는 패턴이 있을 경우, <A>가 샤워하는 이유는 "<A>가 퇴근을 했다."는 패턴에 의한 결과라고 판단하는 근거로 사용됨



(그림 1) Context 생성과정

3.2 패턴

사용자는 주로 반복되는 행동을 수행한다. 일정한 시간대에 하는 행동이나 특정한 작업을 연속해서 수행하거나 반복한다. 과거에 사용자가 행하였던 행동을 학습하여 저장하고 이를 바탕으로 새롭게 발생한 사용자의 행위가 특정 패턴에 속하는 동작인지를 판단한다. 본 논문에서는 과거에 사용자의 행동을 바탕으로 특정시간에 자주 행동하는 시간패턴과 홈네트워크 상에서 연속된 행위의 반복으로 인한 사용자 행동 패턴으로 분류한다.

3.3 장애

장애란 디바이스나 서비스 수행 중에 예외적인 상황이 발생하거나 이들간의 충돌로 인하여 사용자, 서비스 또는 디바이스의 진행을 방해하거나 기능을 충분히 수행하지 못하게 하는 예외적 상황을 뜻한다. 본 논문은 장애를 Fault와 Conflict로 분류하여 관리한다.

Fault는 디바이스를 사용하고자 하는 사용자 또는 서비스가 실행을 명령했지만 디바이스가 반응을 하지 않는 경우이다. Fault가 발생할 때, 네트워크 상황이나 현재 수행중인 서비스의 존재 여부를 확인하여 어떤 종류의 Fault인지 판단한다. Fault의 종류는 디바이스가 동작하지 않는 디바이스 Fault, 네트워크의 과부하로 인해 디바이스가 동작하지 않는 네트워크 Fault, 서비스 모듈의 오류로 인해 서비스가 동작하지 않는 서비스 Fault로 분류한다.

Conflict는 동시에 같은 디바이스에 접근하거나 동일한 디바이스나 서비스에 대해 서로 상충되는 요청들이 전달된 경우이다. 서비스와 서비스 사이의 Conflict는 서비스 실행간에 충돌이 발생한 경우이다. 예를 들어 홈시어터 서비스는 영화 감상 시에 조명을 끄고, 조명관리 시스템은 사용자가 방 안에 있을 때에 조명을 켜려고 할 때이다. 사용자와 서비스 사이의 Conflict는 서비스 실행과 사용자의 요구가 충돌한 경우이다. 예를 들어 습도서비스로 인하여 가습기가 작동 중이고, 사용자 A가 원격에서 댁내의 모든 기기의 Power-OFF를 요청했을 때이다. 사용자와 사용자 사이의 Conflict는 서로 다른 사용자들이 상이한 명령을 내린 경우이다. 사용자 A는 볼륨을 증가 시키고, B는 볼륨을 감소 시킬 때이다.

4. 사용자 패턴 수집 모델

사용자 패턴 수집 모델은 크게 Emulator, 사용자 패턴 수집 서버, 그리고 시각화 도구로 구성된다. Emulator는 서비스 이용 패턴을 찾기 위해 댁내에서 실제로 발생 가능한 Context를 생성하는 툴이다. 이 툴은 다중 사용자 및 다중 서비스 환경을 지원하여, 위치, 접근 가능 사용자 수 등을 디바이스 별로 상세히 설정할 수 있다.

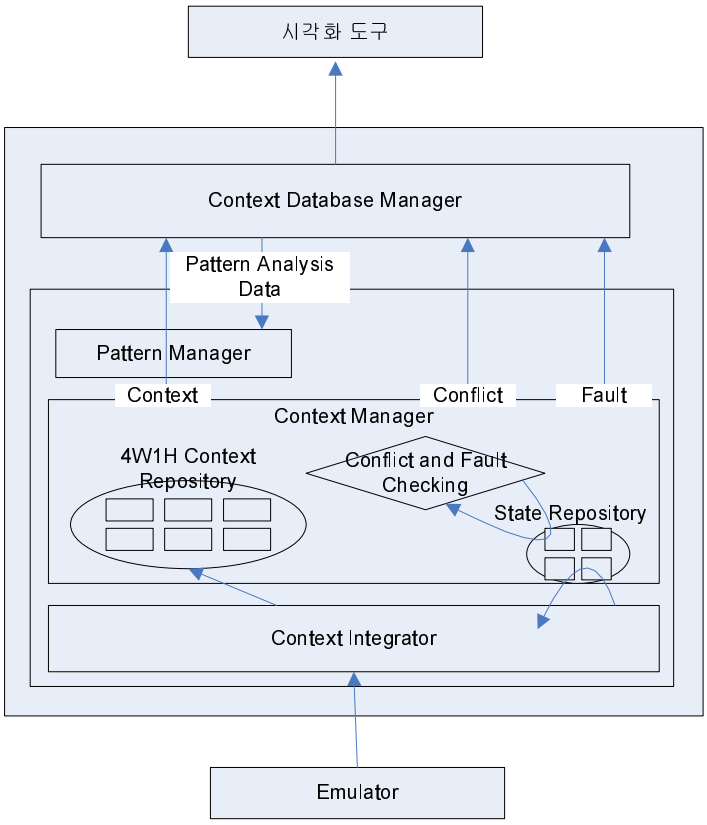
사용자 서비스 이용 패턴 수집 서버는 Emulator로부터 전송되는 불완전한 Context를 서버에서 처리될 수 있도록 의미 있는 Context로 변경하여 저장된다. 서버에서는 댁내 환경에서 정상적인 Context인지 장애인지를 판단하고, 파악된 장애 또는 육하원칙으로 이루어진 Context를 저장하고 관리한다. 이렇게 생성된 Context 정보들을 기반으로 패턴을 분석하여 댁내에서 발생하고 있는 사용자 서비스 이용 패턴을 찾아낸다. 파악된 이용 패턴은 DB server에 저장된다.

시각화 도구는 네트워크, 디바이스 등의 사용자 또는 시간별 사용빈도나 사용 시간 등 다양한 정보들을 그래프를 이용하여 시각화 한다.

4.1 사용자 서비스 이용 수집 서버

사용자 서비스 이용 수집 서버는 Emulator에서 전송하는 Context를 수집하여 사용자의 서비스 이용패턴과 장애를 찾아내는 역할을 한다. Emulator에서 생성하는 불완전한 Context를 통합하여 완전한 Context 생성, 통합 Context에서 사용자의 서비스 이용패턴 생성, Context에서 Conflict, Fault를 탐지, 생성된 패턴, Context, Fault를 DB에 저장 관리한다.

4.1.1 구조

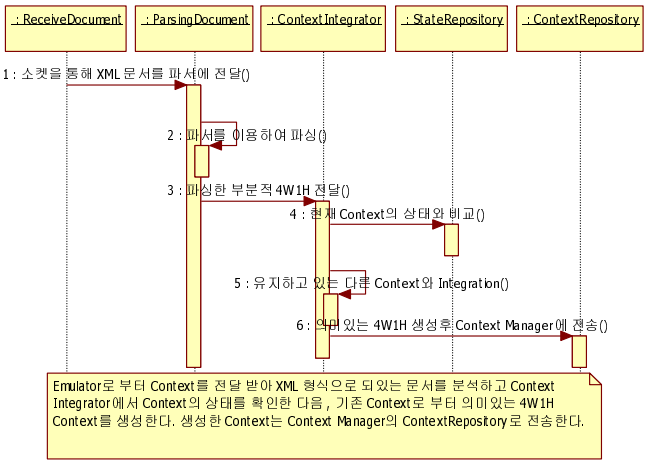


(그림 2) 사용자 서비스 이용패턴 모델 구조

패턴 수집 서버의 주요 모듈은 다음과 같고 구조는 (그림 2)과 같다.

- Context Integrator
- Context Manager
- State Repository
- Pattern Manager
- Context Database Manager

Context Integrator는 단편적인 정보를 가진 Context를 통합하여 댁내의 환경을 전반적으로 표현하는 Context를 생성하는 모듈이다. Emulator에서 생성하는 Context를 받아



(그림 3) Context Integrator 동작과정

기존에 유지하고 있는 다른 Context와 통합을 통해 완전한 Context를 생성하며 이 과정은 (그림 3)와 같다.

Context Manager는 Context 분석을 위해 Context Integrator 모듈이 보내주는 4W1H Context 정보를 DB에 저장하기 전에 분석을 통해 Conflict나 Fault의 발생 여부를 점검하고, Conflict나 Fault의 종류를 분류하는 모듈이다.

State Repository는 댁내의 상황을 파악하기 위해 댁내에 존재하는 장치와 사용자의 상태를 관리하기 위해 이를 저장하는 모듈이다. 이 저장공간에 저장된 자료는 Fault가 일어날 경우를 검출하는 사용된다. Fault는 사용자가 행위를 했음에도 불구하고 디바이스의 상태가 변하지 않는 경우에 발생하기 때문에 사용자의 행위 Context가 전송된 경우 일정시간 후에 상태가 변했는지를 체크하여 Fault임을 인지한다.

Pattern Manager는 기존 DB에 저장되어 있는 Context를 검색한 뒤 각 사용자나 서비스 단위로 이를 정리하여 패턴을 분석하는 모듈이다. 분석된 패턴은 하나의 독립된 Context 또는 Context의 조합으로 존재하며, 이 Context는 기존 Context와의 비교 대상이 된다. 기존 패턴과 현재 발생한 Context가 동일한 경우 Why에 이용 패턴을 삽입함으로써 사용자나 서비스의 디바이스 사용패턴을 쉽게 찾을 수 있도록 한다. 시간패턴은 주 단위 시간 패턴과 일 단위 시간패턴으로 분류하여 관리되며, DB에 저장되어 있는 디바이스 사용시간을 가지고 시간대별 가중치를 구함으로써 패턴을 분석한다. 행동패턴은 주 단위 행동패턴과 일 단위 행동패턴으로 분류하여 관리한다. DB에 저장되어 있는 4W1H의 Context를 가지고 행동의 가중치를 구함으로서 패턴을 분석한다. 기본적으로 순차패턴 알고리즘을 사용한다.

Context Database Manager는 서버와 DB를 분리하기 위하여 DB server와의 연동을 관리하는 모듈이다.

4.1.2 패턴 검색

패턴은 시간 패턴과 행동 패턴으로 분류된다. 시간패턴은 일 단위 패턴과 주 단위 패턴으로 분류하며 시간 패턴을 찾는 방법은 수집된 Context 중에서 사용자나 디바이스 별로

<표 2> 시간 패턴 탐색 리스트

	Who	What	Where	When	How	Count
1	User1	TV	LivingRoom	8	Watch	4
2	User1	TV	LivingRoom	9	Watch	6
3	User1	TV	LivingRoom	10	Watch	4
4	Service2	Boiler	BoilerRoom	19	Power.On	3
5	Service2	Boiler	BoilerRoom	20	Power.On	5
6	Service2	Boiler	BoilerRoom	21	Power.On	4

<표 3> 일 단위 시간패턴 리스트

Pattern Name	Who	What	Where	Start	Start	End
User1_D	User1	TV	Living -Room	Watch	8	10
Service2_D	Service2	Boiler	Boiler -Room	Power.On	19	21

발생빈도가 높은 Context를 찾는 것이다. 예를 들어 <표 2>는 DB에 저장된 Context의 일부이다. User1의 경우 8시~10까지 TV를 시청하는 것이 가중치가 높기 때문에 User1의 시간패턴이 된다. 또한 Service2가 19~21시까지 보일러를 작동하는 것이 가중치가 높기 때문에 Service2의 시간 패턴이 된다. 이에 따라 저장되는 시간 패턴은 <표 3>과 같다.

행동 패턴은 일 단위 행동패턴과 일 단위 행동패턴으로 분류하며, DB에 저장되어 있는 4W1H Context를 이용하여 행동의 가중치를 구하여 패턴을 발견한다. 행동패턴은 특정 사용자가 일련의 행동을 순차적으로 수행한 것으로 정의한다. 기본적으로 순차패턴 알고리즘을 사용하며 부분적인 오차를 허용한다.

4.2 Emulator

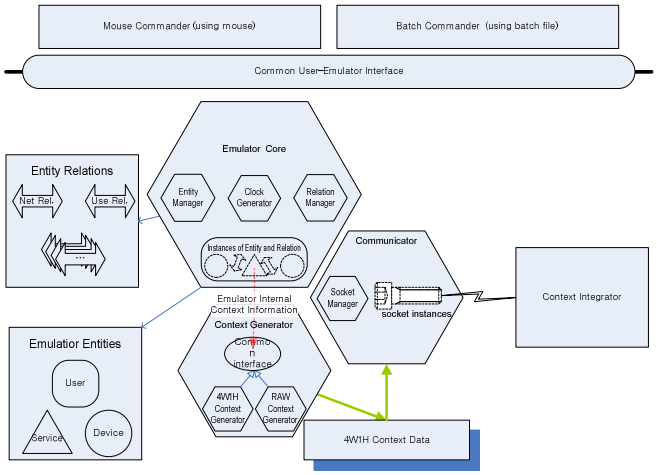
Emulator는 본 논문에서 사용자의 서비스 이용패턴 수집에 사용하는 부분적인 Context를 생성하는 테스트베드용 툴이다. 따라서, 실존하는 인텔리전트 홈 환경과 유사한 사용형태와 구성요소를 가진다. Emulator에 대한 요구사항은 다음과 같다. 패턴수집에서 사용할 불완전한 4W1H Context를 생성, 다중 사용자 및 다중 서비스 환경 지원, 마우스 및 배치파일을 이용한 Context 생성. Fault 상황을 확률적으로 반영한 Context 생성한다.

4.2.1 구조

Emulator 전체 구조는 (그림 4)와 같고 주요 모듈은 다음과 같다.

- Emulator Core
- Context Generator
- Communicator

- Emulator Entities
- Entity Relations
- Common User-Emulator Interface
- Batch Commander



(그림 4) Emulator의 구조도

Emulator Core는 Emulator의 각 요소들을 관리하여 동작하도록 하는 부분으로서 Clock Generator, Entity Manager, Relation Manager, Instances of Entity and Relation등으로 구성된다.

Context Generator는 Emulator가 동작하면서 발생시키는 데이터를 부분적인 4W1H Context 형식으로 변환한다. 향후에 부분적 4W1H 형식 요구사항 변경에 대해서는 인터페이스를 상속하여 구현함으로써 이를 통해 실제로 Emulator가 UPnP, Jini, Zigbee등의 디바이스의 Context 생성 가능하도록 한다. Communicator는 Context Integrator 모듈과 Emulator가 생성한 Context XML 정보의 통신 및 교환을 담당한다.

Emulator Entities는 Emulator의 Emulation 대상이 되는 구성요소, 사용자, 디바이스, 서비스에 의해 정의된 클래스이다. Entity Relations는 Emulator의 개별 구성 요소들간의 상관 관계를 정의한 클래스들의 집합이다.

Common User-Emulator Interface는 사용자와 Emulator 간 동작에 대해 공통된 인터페이스를 제공한다.

Batch Commander는 Common User-Emulator Interface를 통해 사용자가 만들어둔 배치 파일을 분석하여 Emulator를 동작시킨다.

3.2.2 배치파일

Emulator는 GUI의 입력과 배치파일의 입력으로 Context를 생성한다. 일련의 동작을 배치파일로 작성을 하여 Emulator에 입력하면 이를 해석하여 Context를 생성한다. 배치 파일은 한 개 이상의 배치 아이템으로 이루어지며, 각 아이템은 해당 액션의 행동간격, 행동주체, 행동, 대상장비 등의 4가지 요소로 구성된다. 배치파일 작성에서 사용하는 양식은 occurrence clock : action-source : action : action-data(target-name)의 형태이다. 배치파일의 예는 <표 4>와 같다.

<표 4> 배치파일의 예

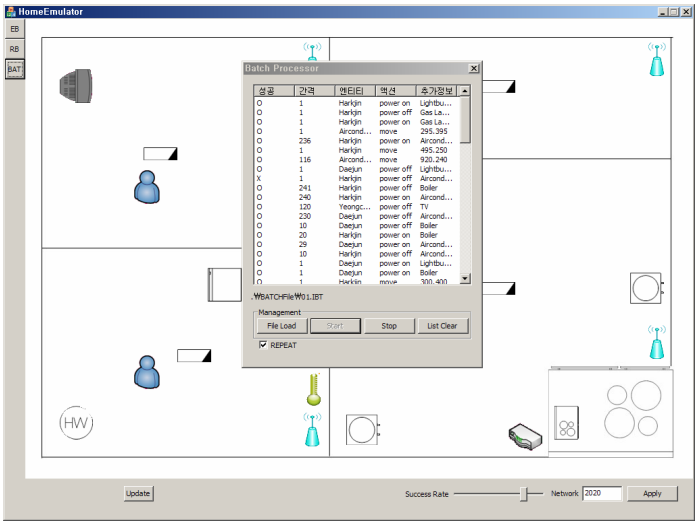
```
# "행동간격:행동주체:행동:대상장비"
# 배치는 상기와 같은 형태로 구성됨
236:Harkjin:Power On:Airconditioner
1:Airconditioner:Move:295.395
1:Harkjin:Power On:Gas Lange
1:Harkjin:Power Off:Gas Lange
1:Harkjin:Power On:Lightbulb2
```

4. 테스트

전체 테스트 구조는 Emulator에서 Context를 생성하며 이를 사용자 패턴 수집 서버로 전송한다. 서버에서 생성되는 패턴, Fault는 시각화 도구를 이용하여 검색한다.

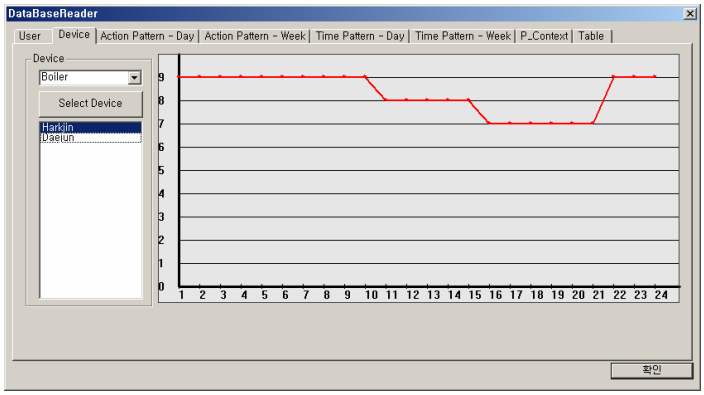
Emulator의 개발환경은 Window 2003이며 컴파일러는 Visual studio 2003 개발언어는 C++이며 테스트에 설정된 기기는 TV, 에어컨, 냉장고, 라우터 등의 디바이스와 위치, 온도, 습도 등의 센서와 다중의 사용자를 설정하였다. 서버와의 Context 교환은 각각의 디바이스, 센서, 사용자별로 소켓을 이용한다. Emulator는 GUI와 배치 파일을 이용한 두 가지 방식으로 동작 한다. Context를 자동으로 생성하기 위해서는 배치 파일이 필요하며 내용은 <표 2>처럼 작성한다. 프로그램 실행 시에 배치 파일을 호출하여 자동으로 Context를 생성할 수 있다.

Emulator가 서버에 접속되면 온도, 습도, 위치 센서는 주기적으로 Context를 생성한다. 각각의 디바이스는 상태가 변경될 때마다 상태를 알려주는 Context를 생성한다. 생성된 Context는 로그 창을 통해 확인할 수 있다. 패턴 수집 서버의 개발환경은 Linux 2.6이며 컴파일러는 g++ 언어는 C++를 사용하였다. 수집 서버는 콘솔 창으로 Emulator에서 전송된 Context를 출력한다. (그림 5)은 배치파일을 이용하여 Context를 생성하는 Emulator의 모습이다.

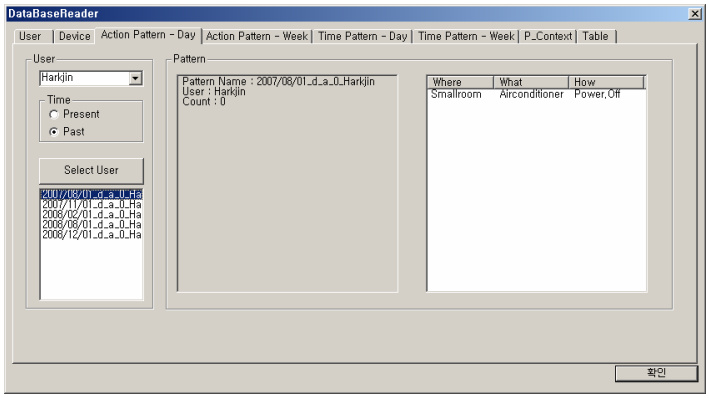


(그림 5) 배치파일을 이용한 Context 생성

패턴 수집 서버에서 찾아낸 Context, 패턴, Fault는 시각화 도구를 이용해서 (그림 6,7)과 같이 사용자 별로 확인 할수 있다.



(그림 6) 사용자 별 시간패턴



(그림 7) 사용자 별 행동패턴

5. 결론

그 동안 홈네트워크를 위한 이 기종 미들웨어 간의 연동에 관한 많은 연구가 이루어졌다. 하지만 단순한 연동은 사용자들의 유비쿼터스에 대한 기대치를 충족하지 못한다. 홈네트워크 미들웨어의 연동을 통해 홈네트워크를 비롯한 유비쿼터스가 효율적으로 이용되기 위해서는 설정이나 운영이 쉬워야 한다. 그 중에 하나로 댁내에서 예상치 못한 문제가 발생하거나 고장이 발생한 경우 이를 처리할 수 있는 고장 감내의 기능이 필요하다. 고장을 처리하기 이해서는 우선 고장을 인지할 수 있어야 한다. 이를 위해 댁내에서 발생하는 Context를 기반으로 고장을 발견하고 이를 처리하는 시스템을 개발해야 한다. 그 다음으로는 고장을 예측을 해서 고장이 발생하기 전에 사용자에게 경고나 시스템 스스로 처리하도록 해야 한다.

본 논문에서는 댁내에 존재할 수 있는 다양한 종류의 Context를 정하고 이들 간의 관계에 따라 Context를 발생하는 Emulator를 구현하였다. 그리고 Emulator에서 발생한 불완전한 Context를 통합하여 완전한 Context를 생성한다. 또한 생성된 Context를 통해서 사용자의 서비스 이용패턴, Fault, Conflict를 찾는 서버를 구현했다. 이를 통해 댁내에서 발생하는 상황을 파악하고 Fault와 Conflict를 검출하여 고장을 발견했다. 또한 사용자의 서비스 이용패턴을 통해 고장 원인을 찾을 때 분석할 수 있는 데이터를 생성했다. 마지막으로 시각화 도구를 통해서 패턴 수집서버에서 찾아낸 패턴, Fault, conflict, DB table을 확인 할 수 있다.

향후에는 현실세계의 다양한 디바이스와 센서, 사용자간의 영향을 주는 상관관계를 분석해서 Emulator가 생성하는 Context의 신뢰도를 높이는 연구가 필요하다. 또한 패턴을 찾는 알고리즘의 성능 향상과 본 논문을 바탕으로 고장이 발생한 원인을 찾는 것이 아닌 고장의 발생을 예측하고 미리 처리할 수 있는 기법에 대한 연구가 요구된다.

참고문헌

[1] Georgia Institute of Tech., “Aware Home”, Http://www.awarehoem.gatech.edu

[2] S.Jang and W.Woo, “ubi-UCAM:a Unified Context-Aware Application Model”, Lecture Note Artificial Intelligence, Vol, 2680, pp.178-189, 2003

[3] T. Zimmer, “Towards a Better Understanding of Context Attributes.”, Proc. Of 2nd IEE Int’l Conf. on Pervasive Computing and Communications, pp.23-28,2004

[4] P.Gray, D.Salber, “Modeling and using sensed context in the design of interactive applications”, In Proceedings of 8th IFIP Conference on Engineering for Human-Computer Interaction, Toronto, 2001

[5] Dey, A.K. Abord, G.D “Toward a Better Understanding of Context and Context-Awareness.”, CHI 2000 Workshop on the What, Who, Where, When and How of Context-Awareness, 2000

[6] M.Kim and S. Kim, “A Scenario-Based user-Oriented Integrated Architecture for Supporting InterOperability Among Heterogeneous Home network Middlewares”, ICCA2006, pp669-678, 2006

[7] Hong, J. I.,et al. “An Infrasturcture Approach to Context-Aware Computing.”, Human-Computer Interaction(HCI) Journal, Vol16, 2001

[8] Schilit, B., Adams, N.,and Want, R. “Context-Aware Computing applications.”, IEEE Workshop on Mobile computing systems and Applications, Santa Cruz, CA, Us, 1994