韓國 컴퓨터情報學會 論文誌
第12卷 第1號, 2007. 3.

2007-12-1-1-12

통합 상황 프로비저닝 전략을 기반으로 한 모바일 폰 미들웨어의 설계

정 현 진*, 원 유 헌**

Design of Mobile Phone Middleware based on Integrated Context Provisioning Strategy

Hyun-Jin Jeong*, You-Hun Won**

요 약

근래들어 PDA나 스마트 폰과 같은 모바일 기기에서 동작하는 어플리케이션에서 상황 정보를 이용하게 만드는 것은 유비쿼터스 컴퓨팅, 모바일 컴퓨팅 분야에서 관심을 가지게 되었다. 기존 미들웨어들은 대부분 상황 프로비저닝을 위해 한 가지 전략을 사용하였다. 그러나, 본 논문에서 제안한 미들웨어는 상황 프로비저닝을 위해 내부 센서 기반, 외부 인프라스트럭처 기반, 에드-혹 네트워크에서의 분산 기반 전략을 통합하였다. 어플리케이션은 필요로 하는 상황 아이템을 SQL 형태의 상황 질의어를 사용하여 요청하며 자원의 이용도나 외부 인프라스트럭처의 존재 유무 등이 고려된 상황 프로비저닝 전략을 사용하여 상황 정보를 획득하는 것이 가능하다.

Abstract

In these days, the use of context in application running on mobile devices such as PDAs and smart phone has become a crucial requirement for several research areas, including ubiquitous computing, mobile computing. Previous middlewares which support context provisioning uses single strategy. But, this paper proposed middleware integrated multiple strategies for context provisioning, namely internal sensors-based, external infrastructue-based, and distributed provisioning in ad hoc networks. Applications can query needed context items using SQL like context query language and require context information to use different provisioning mechanisms depending on resource availability and presence of external infrastructures.

▸ Keyword : 상황정보(context-information), 미들웨어(middleware), 모바일(mobile)

• 제1저자 : 정현진
• 접수일 : 2007.2.25, 심사일 : 2007.3.5, 심사완료일 : 2007. 3.25.
* 홍익대학교 컴퓨터공학과 박사 수료, ** 홍익대학교 컴퓨터공학과 교수
※ 이 논문은 2004학년도 홍익대학교 교내연구비에 의하여 지원되었음(This work was supported by 2004 Hongik University Research Fund).

I. 서 론

PDA나 스마트 폰과 같은 모바일 기기에서 동작하는 어플리케이션에서 상황(context)의 사용에 대한 연구는 지난 10여 년간 유비쿼터스 컴퓨팅이나 모바일 컴퓨팅과 같은 연구 분야의 주요 관심사 중 하나였다. 모바일 기기를 사용하는 사용자는 한곳에 고정되어 있지 않고 계속 빈번하게 이동하기 때문에 개인 맞춤화 서비스나 적응형 서비스에서 이용하고자 하는 상황은 빈번하게 변하게 되므로 시스템에서는 좀 더 효율적으로 상황을 접근해야 할 방법이 필요하다.

상황 정보는 모바일 기기내에 장착된 센서들 또는 주변 환경에 내재된 다수의 센서, 카메라 또는 마이크로 폰을 사용하여 획득할 수 있다. 그러나, 모바일 기기에서 실제로 상황 정보를 획득하기 위해서 고려해야 할 여러 가지 문제가 있으며 이를 정리하면 다음과 같다[1].

첫째, 상황 정보는 다수의 서브 프로세스들로 이루어지는 복잡한 처리 과정을 거쳐서 얻을 수 있는데 이 처리 과정에서 다량의 전원 소비를 유발한다. 현재 모바일 기기들의 전원은 대부분 배터리를 통해 공급받고 있으며 배터리의 기술적 한계로 인하여 사용 시간에 제약을 받고 있다. 이런 환경에서 다량의 전원 소비를 유발하는 상황 정보를 처리한다는 것은 그만큼 사용 시간이 짧아질 수 있다는 것이므로 부담스러운게 사실이다. 특히 고급 상황 정보를 얻기 위한 추론 알고리즘의 처리시에는 보다 많은 메모리 공간과 고성능의 컴퓨팅 파워를 필요로 한다. 둘째, 기기에 내장하게 될 센서들의 이동성, 사용성, 수명, 비용 등을 고려해야 한다. 만약 선세의 크기가 커서 기기에 내장할 수 없다면 사용성이 떨어지게 되는 것이고 기기의 가격보다 센서 하나의 가격이 비싸다면 역시 문제가 발생할 수 있다. 마지막으로 GPS 장치는 실내에서 작동하지 않는 것처럼 어떤 특별한 환경에서는 센서가 작동하지 않을 수 있다. 이런 이유들로 인해 모바일 기기에서는 어떻게 상황 데이터를 모으고 적절하게 처리하여 상황 정보를 만드느냐는 적절한 전략을 세우는 것이 중요한 설계의 요소가 된다.

일반적으로 모바일 기기에서 상황 정보를 획득하는 전략은 크게 세 가지 정도로 나눠볼 수 있다. 첫째, 상황 인식 어플리케이션이 직접 센서로부터 상황 정보를 획득하거나[6] 또는 지역적으로 상황 데이터를 처리하여 획득하는 방법을 사용한다[7]. 둘째, 외부의 상황 인프라스트럭처에 의존한다[7][13]. 셋째, 분산 환경의 모바일 에드-혹 네트워크상에서 서로 다른 유형의 상황 정보를 공유한다.

세 가지 전략 메커니즘은 모두 해당되는 특정 환경에서만 유용하다. 예를 들어, 외부 인프라스트럭처에 의존하는 상황 인식 어플리케이션은 의존하고 있는 인프라스트럭처를 이용하여 상황 정보를 획득하도록 개발된다. 만약 통신 두절이나 고장으로 인해 인프라스트럭처를 이용하지 못하는 경우가 발생할 경우엔 자신의 센서나 프로세서를 이용하거나 연결된 이웃 기기의 센서를 사용해야 한다.

본 논문에서 제안하는 미들웨어는 모바일 폰에서 다수의 상황 프로비저닝을 지원할 수 있도록 설계하였다. 기존의 방법들은 한가지 특정 환경만을 고려한 상황 프로비저닝 전략을 사용할 수 있도록 만들어진 것을 탈피하여 본 논문의 미들웨어는 다양한 프로비저닝 전략을 환경에 따라 가변적으로 적용할 수 있다. 이렇게 함으로써 특정 프로비저닝 전략을 사용할 수 없게 되었을 경우 다른 프로비저닝 전략을 적용하여 끊김 없이 상황 정보를 제공할 수 있다. 이를 위해 제안한 미들웨어에서는 먼저 상황 인식 어플리케이션이 자신이 필요로 하는 상황 데이터를 SQL 형태의 질의어를 사용하여 원하는 상황 데이터를 질의하도록 하였다. 질의의 처리는 질의어 처리기가 담당하며 질의어 처리기에서는 질의를 분석하여 어플리케이션에서 원하는 상황 데이터를 제공할 수 있는 상황 프로비저닝 전략을 선택한다. 선택한 프로비저닝 전략을 가동하여 수집한 상황 데이터는 해당 어플리케이션에게 전달된다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 상황 프로비저닝과 관계된 연구를 소개한다. 3장에서는 본 논문에서 제안한 미들웨어 구조를 설명한다. 4장에서는 제안한 미들웨어 구조의 실험을 통해 가능성을 검증하고 마지막으로 5장에서는 결론과 향후 연구 과제를 제시한다.

II. 관련 연구

2.1 상황 프로비저닝 전략

상황이란 현실세계에 존재하는 객체(entity)의 상태를 추상화하여 만든 정보라고 정의 할 수 있다. 즉, 객체의 상태를 정보화한 것으로 간략화 해 볼 수 있다. 여기서 객체

란 사람, 사물, 장소 또는 사람과 서비스와의 상호작용을 모두 포함한 것을 의미한다[2]. Dey는 상황을 어플리케이션과 사용자 사이의 객체 상태를 나타내는 정보라고 정의 하였다[3]. Jang은 5W1H(Who, What, Where, When, Why, How)[4]로 정의하였으며 국내 광주과학기술원의 우운택의 연구에서는 이를 조합하여 상황 정보를 정의하였다.

상황 정보를 필요로 하는 어플리케이션이나 다른 모듈에게 상황 정보를 생성하고 공급하는 일련의 처리 과정을 상황 프로비저닝이라고 하며 이를 담당하는 소프트웨어 컴포넌트를 상황 제공자(context provider)라고 한다. 상황 정보는 가공되지 않은 상황 데이터의 수집으로부터 시작되며 이런 상황 데이터는 기기에 내장된 센서, 태그, 위치 시스템, 바이오센서 등을 통해 수집된다. 상황 제공자는 수집한 가공하지 않은 상황 데이터를 특징 추출(feature extraction), 통합(aggregation), 분류(classification), 클러스터링과 같은 기법을 사용하여 보다 높은 수준의 상황 정보로 만든다. 최종적으로 만들어진 상황 정보는 어플리케이션이나 다른 외부 컴포넌트에공급된다. <그림 1>은 상황 데이터의 공급 체인 구조를 나타낸 것이다.

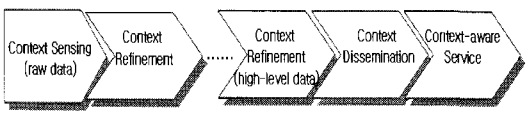

그림 1. 상황 프로비저닝 공급 체인
Fig 1. The Supply chain of context provisioning

모바일 기기에서 상황 프로비저닝을 위한 기존 연구들은 하나의 전략에 초점을 맞춘 경우가 대부분이며 내부 센서 기반 전략, 외부 상황 인프라스트럭처 기반 전략, 분산 기반 전략중 하나를 이용한다.

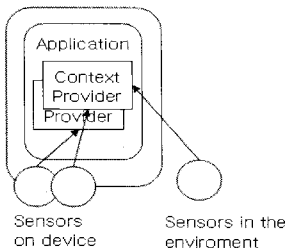

그림 2. 내부 상황 프로비저닝
Fig 2. Internal context provisioning

첫 번째 내부 센서 기반 전략에서는 기기내에 설치된 특수 목적의 상황 제공자를 두고 상황 프로비저닝을 수행한다. <그림 2>는 내부 상황 프로비저닝 전략을 나타낸다. 이 전략에서 상황 제공자와 어플리케이션은 서로 통합된 구조로 구성되기 때문에 어플리케이션 개발이 어렵고 재사용이 힘들며 일반성이 떨어지는 문제점이 있다. 이런 문제를 해결하기 위해 상황 툴킷[8]에서는 상황 제공자를 위젯이라 불리는 컴포넌트로 정의하여 어플리케이션과 상황제공자를 분리하였다. 상황 툴킷은 위젯이 센서로부터 상황 데이터를 읽고 상황을 수집하는 모듈로 전송하는 구조로 설계되어 있다. <그림 3>은 상황 툴킷의 컴포넌트 구조이다.

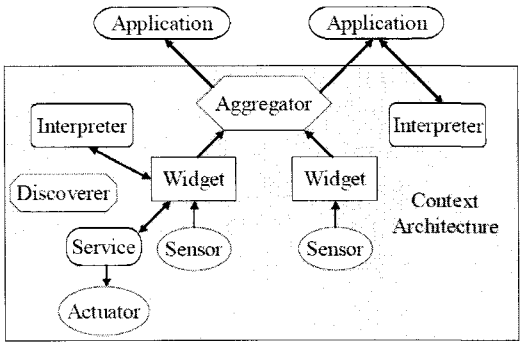

그림 3. 상황 툴킷 컴포넌트 구조
Fig 3. Architecture of Context Tookit Component

상황 제공자를 프레임워크 구조로 설계(TEA[5])하거나 미들웨어 구조로 설계(RCSM[9])하기도 한다. 이러한 구조에서는 어플리케이션 개발자가 동일한 상황 추상화를 제공할 수 있게 해주는 장점이 있으나 모바일 기기가 어떠한 형태의 센서를 가지고 있다는 가정하에서 가능한 구조라는 문제점이 있다.

두 번째 전략은 어떤 어플리케이션에서도 접근할 수 있는 별도의 기기를 두고 이 기기에서 프로비저닝을 수행하는 컴포넌트를 통해 상황 프로비저닝을 수행하는 방법이다. 이런 전략을 외부 중앙집중형 상황 프로비저닝이라고 하며 <그림 4>는 외부 중앙 집중형 상황 프로비저닝을 나타낸다.

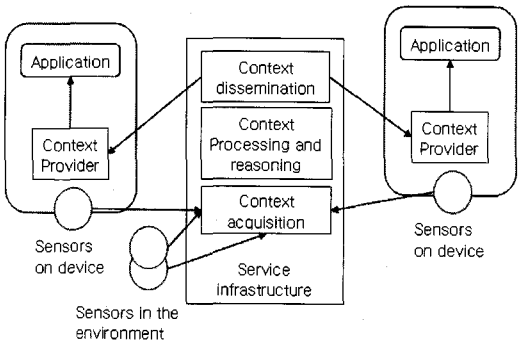

그림 4. 외부 중앙집중형 상황 프로비저닝
Fig 4. External centralized context provisioning

상황 프로비저닝을 수행하는 원격 기기상의 컴포넌트구성은 Confab[10]과 같이 외부 인프라스트럭처 형태로 하거나 AT&T의 연구[11]처럼 서버를 공유하는 형태로 만들 수 있다. AT&T의 연구[11]는 일정 공간내에 있는 사람의 위치를 파악하는 프로젝트이다. 공간내에 있는 사람의 위치는 센서(Bat Sensor)로부터의 데이터가 공간 인덱싱 프락시(Spatial Indexing Proxy)서버에 전달되고 이를 처리하여 모바일 어플리케이션에 전달하는 구조로 설계되어 있다. 〈그림 4〉는 AT&T의 구조이다.

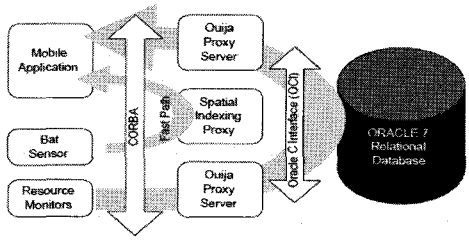

그림 5. AT&T의 시스템 구조
Fig 5. System Architecture in AT&T

외부 중앙집중형 상황 프로비저닝 전략에서는 필요로 하는 상황 데이터를 어떤 센서가 제공하는지를 찾는일부터 이를 처리하고 저장하고 배포하는 일을 외부의 서비스 인프라스트럭처에 의존한다. 서비스를 사용하고자 하는 기기들의 어플리케이션은 인프라스트럭처의 해당 모듈을 공유하여 서비스를 받는다. 이 전략은 여러 가지 장점을 갖는다. 첫째, 여러 기기가 센서, 자원, 프로세서를 공유하므로 각 기기의 처리 부담을 현저히 줄일수 있다. 둘째, 어플리케이션이 특정 센서 플랫폼과 묶일 필요가 없으며 어떤 네트워크 장치상에도 실행될 수 있다. 셋째, 중앙 서버에 상황 데이터를 모으고 공유하므로 어플리케이션은 자신이 접근할 수 없는 곳의 상황 데이터도 이용할 수 있다. 그러나, 모든 상황 프로비저닝을 한곳에 의존해야 하므로 확장하기 어렵고 중앙 서버의 오류나 고장시 문제가 발생할 수 있다.

세 번째 전략은 분산 처리 네트워크를 기반으로 한 것으로 실제 적용된 사례가 드문 전략이다. 분산 기반 전략의 기본적인 아이디어는 분산 데이터베이스에서 데이터에 접근하는 것과 유사하게 상황 프로비저닝을 처리한다. 각 기기는 에드-혹 네트워크로 연결되어 있고 각 기기에서 동작하는 상황 제공자를 노드로 간주하고 통신을 통해 각 상황 제공자별로 처리한 상황데이터를 주고 받는다. 〈그림 6〉은 분산 기반 전략을 나타낸 것이다.

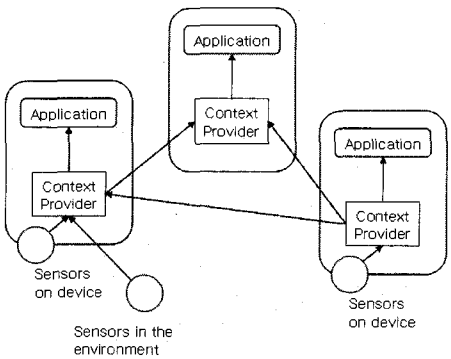

그림 6. 외부 분산 상황 프로비저닝
Fig 6. External distributed context provisioning

이 전략에서는 인접한 위치에 있는 기기들끼리 유사한 상황 뷰를 공유할 수 있다. 센서를 가진 기기는 가공되지 않은 상황 데이터를 획득하고 처리하며 이를 이웃한 다른 기기들이 이용할 수 있다. 또한 일련의 상황 인식 기기들을 그룹으로 묶어 서로 협력하여 상황 인식 처리를 하는 것도 가능하다.

본 논문에서 제안한 미들웨어에서는 앞에서 기술한 세 가지 프로비저닝 전략을 통합한다. 미들웨어에서는 상황 인식 어플리케이션의 요구사항과 기기에서 이용 가능한 자원 현황을 기반으로 가장 적합하다고 여겨지는 프로비저닝 전략을 채택하고 처리한다.

III. 통합 상황 프로비저닝을 적용한 모바일 폰 미들웨어의 설계

본 장에서는 본 논문에서 제안한 여러 상황 프로비저닝 전략을 통합한 모바일 폰 미들웨어의 구조에 대해 설명한다. 미들웨어는 어플리케이션으로부터 받은 상황 질의 요청을 상황 생성(Context Production) 모듈내의 질의 처리기에 전달하여 해석한 후 현 시스템의 자원과 상황 제공자의 현황을 고려하여 적합한 상황 원천으로부터 상황 데이터를 제공받은 후 이를 요청한 어플리케이션에 전달하는 구조로 이루어진다. 전체적인 미들웨어의 개념 구조는 〈그림 7〉과 같다.

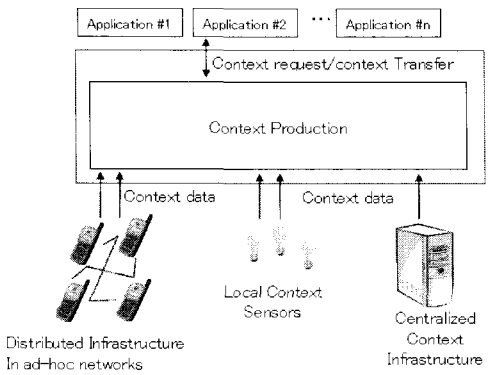

그림 7. 개념적인 구조
Fig 7. Conceptual architecture

상황 데이터의 원천은 프로비저닝 전략에 따라 내부 센서가 될 수도 있거나 에드-혹 기반에서 동작중인 다른 기기 또는 외부 인프라스트럭처가 될 수 있다. 제안한 미들웨어는 다음과 같은 요소를 반영하여 설계되었다.

- 여러 상황 프로비저닝 전략의 통합 사용: 내부 기반 전략, 외부 중앙 집중형 전략, 분산 전략을 모두 지원한다.
- 상시 이용 가능성 : 모바일 환경에서의 상황 프로비저닝은 갑작스런 환경의 변화를 고려해야 한다. 이론적으로 하드웨어의 고장이나 센서와의 통신 끊김 등의 현상이 발생했을 경우에 중단(interruption)없이 계속 동작하여야 한다. 만약, 환경 변화로 인해 특정 프로비저닝 전략을 사용할 수 없을 경우엔 즉시 다른 가능한 대체 전략을 실행한다.
- SQL 형태의 질의어 인터페이스 : 어플리케이션이 필요로 하는 상황 데이터를 요청할 수 있도록 SQL 형태의 상황 질의어를 제공한다. 질의어에서는 어플리케이션이 필요로 하는 상황 데이터의 유형과 속성을 표현함으로써 상황 프로비저닝 전략을 결정할 수 있는 기반 정보로 활용한다.
- 모듈화와 확장성 : 미들웨어의 전체 구조를 모듈화하여 새로운 형태의 센서 개발이나 상황 프로비저닝 알고리즘의 등장을 대비한다.

3.1 상황 아이템과 상황 메타데이터

어플리케이션에서 인식 가능한 상황 정보의 표현은 상황 아이템 집합을 사용한다. 예를 들어 (time, duration)이란 상황 아이템의 집합이 시간 정보를 나타내는 식이다. 또 다른 예를 든다면 사용자 특성 정보는 (activity, mood)로 시간 정보는 (time, duration)로 기기에서 이용 가능한 자원 정보는 (nearby devices, device status)와 같이 나타낸다.

각 상황 아이템은 메타데이터를 사용하여 상황 아이템이 얼마나 최근의 것인지, 얼마나 오랜 기간 동안 유효한 값인지, 얼마나 정확한 값인지, 얼마만큼의 신뢰도를 갖춘 것인지 등을 분류한다.

본 논문에서 제안한 미들웨어에서는 상황 데이터를 상황 아이템 객체로 정의하며 상황 아이템 객체는 type, value, timestamp, lifetime, source, options 필드를 포함한다. 각 필드의 의미는 다음과 같다.

- type : 상황 아이템의 분류
- value : 상황 아이템의 현재 값
- source : 센서의 유형, 인프라스트럭처 ID, 기기 이름과 같은 상황 정보의 원천
- option : 상황 데이터의 추가적인 속성

이 중 source와 options은 선택적인 필드이다.

3.2 상황 질의어(Context Query Language)

어플리케이션이 필요로 하는 상황 데이터를 요청하기 위해 상황 질의어를 사용한다. 상황 질의어는 SQL 형태로 정의하였으며 〈표 1〉과 같은 구성을 갖는다.

표 4. 상황 질의어의 구성
Table 1. Context Query Language Format

```
SELECT {context name} [*]
FROM {source}
WHERE {predicate}
FRESHNESS = time
DURATION = time [*]
EVERY = time
EVENT {predicate}
```

각 절에서 *는 생략 불가능한 절을 나타내는 것이다. SELECT절은 필요한 상황 아이템을 지정한다. FROM절은 수집해야 할 상황 데이터를 제공할 원천의 유형과 특성을 지정하는 것으로 유형은 상황 프로비저닝 전략에서 기술한 세 가지 유형중 하나가 된다. 어플리케이션은 에드-혹 네트워크 기반의 분산 환경에서 동작중인 다른 기기의 상황 아이템을 획득할 수 있는데 이는 FROM절의 destID 필드를 이용하여 지정한 기기에서 상황 아이템을 전송하는 방식으로 이루어진다. WHERE절은 상황 메타데이터와 일치하는 상황 아이템을 필터링하는데 이용한다. FRESHNESS절은 상황 데이터의 최근성 여부를 명시한며 DURATION절은 질의의 유효 기간을 나타낸다. 마지막의 EVERY절과 EVENT절에서 EVERY절은 주기적으로 처리해야하는 질의를 EVENT절은 이벤트기반의 질의를 지정할 때 사용한다.

〈표 2〉는 간단한 질의의 예를 나타낸 것으로 이 질의를 요청한 어플리케이션은 에드-혹 기반으로 연결된 인접한 기기들로부터 기온값을 전달받는 예이다.

표 5. 질의의 예
Table 2.. Example of query

```
〈query name = "temperature"
  select="temperature"
  〈struct name = "from"〉
    〈String name="typeSrc"〉"ad-hoc"〈/String〉
    〈boolean name="all"〉true〈/boolean〉
    〈int name="range"〉1〈/int〉
  〈/struct〉
  〈struct name="where"〉
    〈double name="accuracy"〉0.2〈/double〉
  〈/struct〉
〈/query〉
```

3.3 미들웨어 구조

본 논문에서 제안한 미들웨어 구조는 〈그림 8〉과 같으며 상황 아이템을 검색하기 위한 방법을 제시하는데 주안점을 두었다.. 미들웨어의 전체 구조는 팩토리 메소드 디자인 패턴[12]을 기반으로 설계 하였다. 미들웨어의 구성은 어플리케이션 관리자(Application Manager), 질의 처리기(Query Manager), 상황 수집기(Context Aggregator), 상황 제공자(Context Provider), 상황 발행기(Context Publisher), 상황 저장소(Context Repository)로 구성된다. 미들웨어의 기능은 크게 어플리케이션 관리 기능, 질의 분석과 처리 기능, 상황 프로비저닝 기능, 상황 아이템의 저장/발송 기능으로 구분된다. 각 기능은 해당 모듈들이 담당하며 이에 대한 설명은 다음과 같다.

어플리케이션 관리자(Application Manager)는 어플리케이션 관리 기능을 담당하는 부분으로 모바일 폰의 어플리케이션 중에서 상황 아이템을 요청한 어플리케이션을 관리하는 상황 아이템을 전달하는 역할을 한다. 어플리케이션 관리자는 질의를 요청한 어플리케이션의 구별을 위해 어플리케이션별로 어플리케이션 ID를 생성하여 부과하고 이 ID값의 리스트를 보관한다. 어플리케이션으로부터 받은 요청 질의는 질의 처리기로 보낸다. 질의 처리기는 어플리케이션 관리자로부터 받은 질의를 분석하고 처리하는 기능을 담당한다. 요청 받은 질의에서 상황 아이템의 유형, 속성, 부가 정보들을 추출한 후 이를 제공할 가장 적합한 상황 원천을 제어할 수 있는 상황 제공자(Context Provider)객체를 인스턴스화 한다. 상황 제공자는 프로비저닝 전략에 따라 Adhoc, Local, Infra의 세 유형으로 구분된다. 상황 제공자의 객체화를 할 때 선택하게 될 상황 제공자는 질의에서 직접 지정한 경우는 해당 상황 제공자를 선택하며 그렇지 않을 경우는 시스템 자원과 이용 가능한 상황 원천을 조사하여 세 가지 유형 중 가장 적합한 상황 제공자를 생성한다. 이렇게 생성된 상황 제공자는 자신이 담당하는 상황 원천으로부터 상황 아이템을 모은 후 이를 상황 수집기로 전송한다. 이 과정에서 상황 제공자는 질의에 포함된 대로 유형, 속성, 옵션 정보를 만족하는 상황 아이템을 수집하고 전달한다. 수집한 상황 아이템을 보다 고급 정보로 만들기 위한 추론 기능이 필요할 경우 상황 저장소를 통하여 외부 인프라스트럭처의 추론 기능을 사용할 수 있다. 그러나, 필요로 하는 상황 아이템의 수준

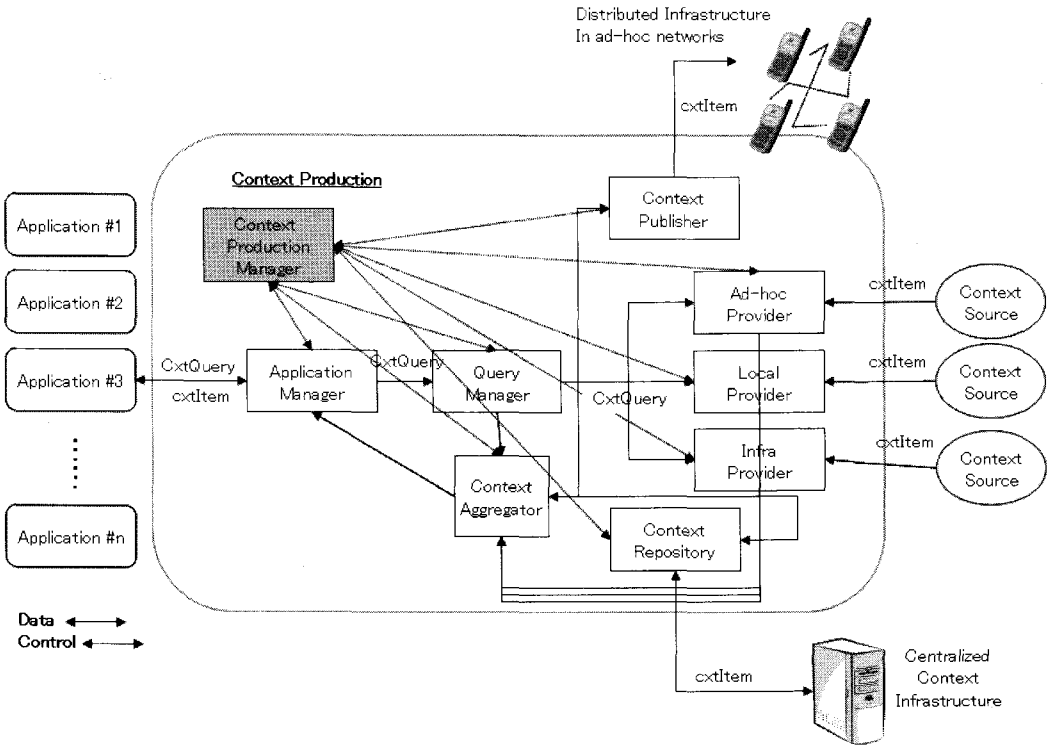

그림 8. 미들웨어의 구조
Fig 8. Structure of middleware

이 어플리케이션마다 각기 다르므로 본 논문에서는 상황 아이템을 추론하는 기능은 해당 어플리케이션이 별도로 처리하거나 별도의 추론 기능 모듈을 둘 수 있는 것으로 가정한다. 현 설계에서의 상황 제공자는 아주 낮은 단계의 추론만 수행 가능한 것으로 한정한다. 상황 발행기(Context Publisher)는 외부 에드-혹 네트워크상에 있는 특정 기기에 상황 아이템을 전송하는 역할을 수행한다. 이를 통해 현재 이용 불가능한 상황 원천의 상황 아이템을 다른 기기로부터 전송 받을 수 있다.

3.4 미들웨어 인터페이스

제안한 미들웨어에서 어플리케이션 개발자를 위해 제공하는 서비스를 인터페이스의 형태로 정의하며 구조는 다음과 같다.

```
public interface ContextProduction {
    String processCxtQuery(CxtQuery q);
    void cancelCxtQuery(String qID);
    String publishCxtItem(String cxtItem);
    void cancelCxtItem(String itemID);
    void storeCxtItem(String cxtItem);
    void registerApplication(String apID);
    String getApplication(String apID);
    Facade makeFacade(CxtQuery qID);
    Facade getFacadeByQueryID(String qID);
    CxtProvider getProviderByQuery(String qID);
    CxtPublisher getCxtPublisher();
}
```

각 메소드 역할은 다음과 같다.

- 질의의 제출 : rocessCxtQuery
- 질의의 삭제 : cancelCxtQuery
- 상황 아이템의 발행 : publishCxtItem
- 상황 아이템의 삭제 : cancelCxtItem
- 상황 아이템의 저장 : storeCxtItem
- 어플리케이션의 등록 : registerApplication

get 메소드는 Facade, CxtProvider, CxtPublisher 객체를 얻는 메소드며 getApplication의 경우는 해당 어플리케이션의 ID값을 얻는 메소드이다.

IV. 응용 시스템의 실험

논문에서 설계한 미들웨어의 실현 가능성을 검증하기 위하여 실험을 진행하였다. 미들웨어는 J2ME CLDC 2.5, MIDP 2.0, CDC 1.0을 기반으로 한 무선툴킷을 사용하여 모바일 폰을 대신한 노트북에 구현하고 설치하여 모바일 폰의 역할을 대신하였다. 실험의 테스트 환경은 〈그림 9〉와 같다.

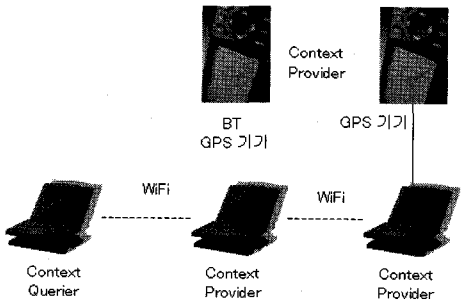

그림 9. 응용 시스템 테스트 환경
Fig 9. Test Environment of Application

〈그림 9〉에서처럼 첫 번째 노트북에는 위치 정보를 필요로 하는 어플리케이션이 설치되어 있다. 위치 정보의 제공은 블루투스를 지원하는 지원하는 GPS기기를 이용한다. 다른 노트북 두 대 중 한대는 블루투스 지원 GPS로부터 위치 정보를 수신하고 다른 한 대는 직접 연결된 GPS 수신기로부터 위치 정보를 획득한다.

실험은 다음과 같이 진행하였다. 상황 질의 노트북에서 필요로 하는 상황 정보인 위치 정보를 블루투스 지원 GPS로부터 획득한다. 이때 블루투스 연결이 단절되었을 경우 이웃 노드인 노트북으로부터 위치 정보를 획득한다. 이웃 노트북과의 연결도 단절되었을 경우 GPS기기와 직접 연결된 노트북으로부터 위치 정보를 획득한다. 이를 통해 환경이 변화하더라도 필요한 상황 정보를 계속 획득할 수 있다는 것을 증명하였다. 모바일 폰에서 동작하는 어플리케이션이므로 필요한 정보의 획득에 걸리는 시간이 길어질 경우는 사용성이 떨어지는 것이므로 수신에 걸리는 시간을 측정하는 방식을 채택하여 실험을 하였다. 〈표 3〉은 실험 결과를 나타낸 것이다. 〈표 3〉에 나타낸 것처럼 상황 아이템의 수집 시간이 이웃한 노드로 이동할 때마다 증가하는 것은 상황 질의를 처리할 때 순서대로 상황 수집 가능 여부를 확인하는 시간이 추가되기 때문이다. 그러나, 해당 환경의 사용이 어렵더라도 다른 경로를 통해 정보를 획득하고 있다. 또한 블루투스의 경우 상황 아이템 수집시간은 WiFi보다 상대적으로 빠르지만 상황 아이템을 생성하는데 걸리는 시간이 길다. 이는 GPS 기기의 성능 때문으로 볼 수 있다.

표 6. 실험 결과
Table 3. Test Result

상황 아이템 발행 지연 시간

전략	평균 지연 시간(ms)
Ad hoc network - BT	140.4
Ad hoc network - WiFi	0.1

상황 아이템 수집 지연 시간

전략	평균 지연 시간(ms)
Ad hoc network - BT	31.8
Ad hoc network - WiFi	761.3 (1 hop)
Ad hoc network - WiFi	14225 (2 hop)

V. 결론 및 향후 연구

본 논문에서는 모바일 폰에서 여러 상황 프로비저닝 전략을 통합한 미들웨어를 제안하였으며 이 연구의 특징을 간략히 요약하면 다음과 같다.

상황 프로비저닝 전략을 사용하는 기존 연구들은 기기 내의 센서기반, 외부 인프라스트럭처 기반, 에드-혹 네트워크의 분산 기반으로 나눠볼 수 있으며 각 전략을 개별적으로 적용한 시스템이 대부분이다. 본 논문에서는 이러한 전략을 통합하여 필요에 따라 적용할 수 있도록SQL 형태의 상황 질의어를 사용하여 어플리케이션이 필요로 하는 상황 아이템을 요구할 수 있도록 설계하였다. 제안한 미들웨어는 어플리케이션 관리기, 상황 질의 처리기, 상황 제

공자의 컴포넌트로 이루어져 있으며 요청된 질의를 분석하여 해당 상황 제공자를 정하고 정해진 상황 제공자가 상황 데이터를 수집하고 이를 다시 어플리케이션에 전달하는 구조로 되어 있다. 본 논문에서 제안한 미들웨어 구조는 기존 연구와 달리 여러 프로비저닝 전략을 통합하여 특정 전략을 지원하는 상황 제공자를 사용할 수 없는 경우라도 다른 전략을 지원하는 상황 제공자로 대체하여 지속적인 상황 정보의 수집이 가능하도록 만들었다는데 의의가 있다. 또한 상황 정보의 수집 모듈과 어플리케이션을 구별함으로써 기능 확장이 필요할 때 해당되는 부분만을 수정할 수 있도록 모듈성을 강화한 것도 특징이다.

향후 연구로는 본 논문에서 제안한 미들웨어를 기반으로 상황 인식 어플리케이션 서비스를 위한 실제 폰 모델로의 적용과 모바일 폰내에서의 추론 기능의 추가, 능동적인 서비스를 위한 상황 질의어의 개선을 모색하는 연구 등이 이루어질 것이다.

참고문헌

[1] A. Shcmidt and K. V. Laerhoven. "How to Build Smart Appliances?", IEEE Personal Communications, Special Issue on Pervasive Computing, 8(4):66-71, August 2001.

[2] Gwizdka, "What's in the context?", Proceedings of Workshop on the What, Who, Where, When, and How of Context-Awareness, Conference on Human Factors in Computing Systems, 2000.

[3] A. K. Dey and G. Abwod, "Towards a Better Understanding of context and context-awareness", Georgia Institute of Technology College of Computing, June 1999.

[4] S. Jang, W. Woo, "5W1H: Unified User-Centric Context", The 7th International Conference on Ubiqitous Computing, 2005.

[5] A. Schmidt, K. A. Adoo, A. Takaluoma, U. Tuomela, K. V. Laerhoven, and W. V. deVelde, "Advanced Interaction in Context", In Proceedings of the First Symposium on Handled and Ubiqitous Computing, p.89-101, Karlsruhe, Germany, September, 1999.

[6] A. K. Dey, D. Salber, and G. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", Human-Computer Interaction, 16(2-4):97-166, 2001.

[7] I. Hong and J. A. Landay, "An Infrastructure approach to context-aware computing", Human-Computer Interaction, 16(2-3):287-303, 2001.

[8] A. K. Dey and G. D. Abowd, "The Context Toolkit: Aiding the Development of Context-Aware Applications", In the Workshop on Software Engineering for Wearable and Pervasive Computing, June 6, 2000.

[9] S. Yau and F. Karim, "A context-sensitive middleware for dynamic integration of mobile devices with network infrastructure", Journal Parallel Distributed Computing, 64(2):301-317, February 2004.

[10] J. Hong and J. Landay, "An Architecture for Privacy Sensitive Ubiqitous Computing", In Proceedings of The Second International Conference on Mobile Systems, Applications, and Services(Mobisys'04), p.177-189, Boston, 2004

[11] A. Harter, A. Hopper, P. Steggles, A. Ward, P. Webster, "The anatomy of a Context-aware applications", Wireless Networks Vol.8, Issue 2/3, pp. 187-197, 2002.

[12] E. Gamma, R. Helm, R. Johnson, and J.Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[13] 김효남, 박용, "유비쿼터스 컴퓨팅 환경에서 상황인식 미들웨어 설계", 한국컴퓨터정보학회 논문지, 제10권 5호, 2005. 11

저 자 소 개

정 현 진

1996년 강남대학교 전자계산학과
1999년 홍익대학교 컴퓨터공학과
2005년 홍익대학교 컴퓨터공학과 박사 수료
2005년 ~ 현재 (주)쏘뉴 연구원
관심분야 : 프로그래밍 언어, 홈네트워킹, 유비쿼터스 컴퓨팅

원 유 헌

1972년 성균관대 수학과
1975년 한국과학기술원 전자계산학과(석사)
1985년 고려대학교 수학과 전산전공(박사)
1976년 ~ 현재 홍익대학교 컴퓨터공학과 교수
관심분야 : 프로그래밍 언어, 컴파일러 이론, 유비쿼터스 컴퓨팅