韓國 컴퓨터情報學會 論文誌
第11卷 第1號, 2006. 3.

2006-11-1-2-5

계층적 상황 온톨로지 관리를 이용한 상황 인식 서비스 미들웨어 설계

이 승 근*, 김 영 민**

Design of Context Awareness Middleware based Hierarchical Context Ontology Management

Seung-Keun Lee*, Young-Min Kim**

요 약

유비쿼터스 컴퓨팅 환경에서는 기존 컴퓨팅 환경에서의 사용자와 컴퓨터간의 대화형 상호작용이 아니라 물리적인 환경, 상황 등을 시스템이 인식하고 이에 따라서 사용자와의 상호 작용을 지원하는 상황 인식 서비스가 중요한 요소로 자리잡고 있다. 상황 인식 서비스는 상황 인식 미들웨어로부터 전달된 상황 정보를 해석할 수 있어야 한다. 기존 연구에서 상황 인식 서비스는 상황 인식 미들웨어에서 사용하는 상황 온톨로지를 이용해서 설계되기 때문에 서비스의 실행도중 상황 온톨로지를 동적으로 변경하기 어렵다. 본 연구에서는 계층적 상황 온톨로지 관리 모델을 제안하고 이를 이용한 상황 인식 서비스 미들웨어를 설계한다. 제안한 모델은 상황 인식 서비스가 실행 중에 필요로 하는 상황 정보를 동적으로 추가할 수 있도록 함으로써 보다 유연하게 상황 인식 서비스를 운용할 수 있도록 지원한다. 또한 상황 온톨로지의 동적인 변화로 인해서 센서로부터 얻은 데이터를 상황 정보로 추론하는 과정에서 발생할 수 있는 상황 모호성(Context Uncertainty)을 해결하기 위해서 상황 충돌 해결 모델을 정의한다. 설계하는 미들웨어는 OSGi 프레임워크 위에서 구현함으로써 다양한 유비쿼터스 환경에 필요한 상황 인식 서비스의 개발 및 운용을 효과적으로 지원을 할 수 있다.

Abstract

The ubiquitous computing environment focuses on recognizing the context and physical entities, whereas, previous computing environments mainly focused on the conversational interactions between the computer and the user. For this reason, there has been an increase in the research of context aware computing environments. In previous researches, context services are designed using context ontology used in context aware middleware. So, context service cannot change the context ontology in execution time. We propose a hierarchical context ontology management for context aware service to change their ontology in execution time. And we also a resolution model for context conflict which is occurred in inference of context. We have designed a middleware based on this model and implemented the middleware. As the middleware is implemented on the OSGi framework, it can cause interoperability among devices such as computers, PDAs, home appliances and sensors. It can also support the development and operation of context aware services, which are required in the ubiquitous computing environment.

▸ Keyword : 상황인식(Context Awareness), 유비쿼터스 컴퓨팅(Ubiquitous Computing), 온톨로지(Ontology)

• 제1저자 : 이승근
• 접수일 : 2006.03.09, 심사완료일 : 2006.03.22
* 인하대학교 컴퓨터정보공학과 강의전담교수, ** 부산경상대학 의료정보과 조교수

Ⅰ. 서 론

지난 50여년간 컴퓨팅 분야에서 사람과 컴퓨터 간의 접근방식은 많은 사람들이 하나의 컴퓨터를 공유하던 방식에서 한 사용자가 하나의 컴퓨터를 사용한다는 개념으로 변화하였으며, 이는 사람들의 컴퓨터 시스템 사용 방식을 크게 바꾸었다. 지난 10여년 동안에 이러한 변화는 더욱 심해져서, 한 사용자가 여러 컴퓨터를 사용하거나 적어도 하나의 컴퓨팅 디바이스를 소유하게 되었으며, 컴퓨터나 장치들은 사용하고 있는지도 모르게 생활 속에서 활용되는 유비쿼터스 컴퓨팅 기술이 다양하게 연구되고 있다[1, 2]. 하지만 유비쿼터스 컴퓨팅은 사용자와 컴퓨터 간의 양적인 관계를 뛰어 넘어 많은 새로운 연구과제들을 제시하고 있다[3, 4]. 특히, 인간이 어떻게 수많은 시스템과 상호 작용할 수 있는지에 관한 이해는 아직 해결되지 않은 상태이며, 현재에도 많은 연구들에서 다루어지고 있다[1,5].

상황(context)은 유비쿼터스 환경 내에서 사용자와 시스템간의 상호 작용을 지원하는 효과적인 정보의 소스로 이용될 수 있다[5]. 물리적인 환경, 상황, 사용자의 역할, 다른 사용자나 환경과의 관계, 사용자의 선호도 및 의도 등은 매우 풍부한 정보의 자원이 될 수 있다. 상황 인식 시스템의 경우 이러한 정보를 사용하여 컴퓨터와의 상호작용을 훨씬 더 쉽게 만들고 심지어는 이러한 명시적인 상호작용을 위한 요구를 제거할 수 있다[6].

많은 스마트 공간을 구성하기 위한 유비쿼터스 컴퓨팅 환경에 대한 연구는 상황 인식 기능을 기본으로 하고 있으며, 이와 관련한 많은 연구들이 여러 대학과 연구 기관에서 진행되고 있다[1,6,10,11]. 일반적으로 상황 인식 서비스는 상황 인식 시스템 또는 미들웨어를 기반으로 하고 있으며, 상황 인식 시스템은 센서로부터 얻어진 데이터를 이용해서 상황 정보를 생성하고 이를 상황 인식 서비스에 전달하는 구조를 갖는다. 상황 인식 서비스들은 상황 인식 시스템에서 전달받은 상황 정보에 따라서 특정 서비스를 사용자에게 제공하게 된다. 따라서 상황 인식 서비스는 상황 인식 시스템으로부터 전달되는 상황 정보가 어떤 의미를 갖는지를 명확히 이해할 수 있어야 한다. 기존 연구에서 상황을 모델링하는 다양한 방법에 대한 연구가 진행되었으나, 온톨로지를 이용해서 상황을 모델링 방법이 시스템간의 의미 교환을 가능하게 하는 장점으로 인해서 많은 주목받고 있다.

그러나 기존 연구에서 진행되는 온톨로지 기반 상황 모델링 방법은 몇가지 문제점을 갖는다. 먼저, 상황 인식 서비스는 설계 단계에서 상황 인식 시스템과 상황 온톨로지를 공유해야 한다. 따라서, 상황 인식 서비스가 시스템에 동적으로 추가되거나 삭제되는 경우 새로운 상황 인식 서비스는 다른 상황 인식 서비스에 영향을 주지 않는 상황에서 시스템과 상황 온톨로지를 공유하지 못한다. 또한, 상황 온톨로지의 동적인 변화로 인해서 센서로부터 얻은 데이터를 상황 정보로 추론하는 과정에서 발생할 수 있는 상황 모호성(Context Uncertainty) 문제를 해결할 수 있어야 한다.

이 연구에서는 계층적 상황 온톨로지 관리 방법에 기반한 상황 인식 미들웨어를 제안한다. 계층적 상황 온톨로지 관리 방법은 상황 인식 서비스와 미들웨어간에 상황 온톨로지를 동적으로 공유할 수 있도록 공통 온톨로지 계층을 설계한다. 상황 인식 서비스는 공통 온톨로지를 이용해서 자신이 필요로 하는 상황 온톨로지를 동적으로 정의하고, 이를 미들웨어에 전달한다. 미들웨어는 상황 인식 서비스로부터 전달받은 상황 온톨로지를 미들웨어에서 관리하는 도메인 온톨로지와 통합하고 이를 관리한다. 상황 인식 미들웨어는 추론되는 모든 상황 정보를 모든 상황 인식 서비스에 전달하지 않고, 상황 인식 서비스가 관심 있어 하는 상황 정보만 전달하도록 한다. 또한, 센서로부터 얻은 데이터가 2개 이상의 상황 정보로 해석되는 상황 모호성 문제를 해결할 수 있는 방법을 제안한다. 제안한 계층적 상황 온톨로지 관리 방법을 이용해서 상황 인식 미들웨어를 설계한다. 설계한 미들웨어는 센서로부터 얻은 데이터를 이용해서 상황 정보를 추론하고, 이 상황 정보를 이벤트 전달 방법을 이용해서 필요로 하는 상황 인식 서비스에 전달한다. 설계한 상황 인식 시스템은 기존 연구에 비해서 서비스의 추가 삭제에 따른 상황 온톨로지의 동적 관리가 가능하고, 상황 모호성 문제를 해결할 수 있는 장점을 갖는다.

Ⅱ. 관련 연구

이 장에서는 기존 연구에서 이루어진 상황에 대한 정의와 모델링 방법에 대해서 기술한다.

2.1 상황 인식

상호 작용에 관한 컴퓨팅 분야에서는 전통적으로 사용자는 일방적으로 컴퓨터에게 직접 정보를 입력하게 하는 메커니즘을 제공한다. 결과적으로 컴퓨터는 인간과 컴퓨터의 대화에서 일어나는 어떠한 상황에 관한 정보도 이용하지 못하고 있다. 컴퓨팅 환경 내의 상황에 관한 이해를 향상시킴으로써, 인간과 컴퓨터의 상호 작용은 보다 풍부한 의사 소통을 가져올 것이며, 좀 더 유용한 서비스를 인간에게 제공하는 것이 가능하게 된다. 그러한 상황을 효과적으로 사용하기 위해서, 먼저 상황이 무엇인지 그리고 그 상황이 어떻게 사용되는지에 관해서 이해해야만 한다.

상황에 관한 이해를 바탕으로, 어플리케이션 개발자는 자신의 어플리케이션 개발에서 어떤 상황을 사용할 지 결정할 수 있게 된다. 따라서 상황이 어떻게 사용될 수 있는지에 관한 이해는 개발자가 자신의 어플리케이션에서 지원하는 상황 인식 행동을 결정할 수 있게 한다.

Schilit와 Theimer는 그들의 연구에서 "상황 인식"이란 용어를 처음으로 사용하였는데, 그 연구에서는 상황을 장소, 사람이나 사물들을 구별 짓는 특징인 아이덴티티(identity), 사람이나 사물들을 포함하는 환경의 변화 등으로 설명한다[7]. Dey는 상황을 사용자가 속해 있는 환경 내에서 사용자의 감정적인 상태, 주의력, 위치와 방향, 날짜와 시간, 사람과 사물 등으로 정의한다[6].

2.2 상황 인식 모델링 방법

상황 모델링은 상황 정보에 대한 높은 수준의 추상적 개념을 제공하기 위해 필요하다. 상황 정보의 다양함과 상황 정보가 쓰이는 도메인 또한 매우 다양하기 때문에, 상황 모델링은 여러 방법을 이용한다. 이러한 상황 모델링 기법은 각 시스템에서의 상황 정보 교환을 위해 사용되는 데이터 구조에 관한 스키마에 의해 분류된다.

(1) 키-값 모델

키-값(Key-Value) 모델은 상황 정보를 모델링하기 위한 가장 간단한 데이터 구조이다. Schilit외[7] 등의 연구에서, 환경 변수로서 상황 정보(예, 위치 정보)의 수치(value)를 어플리케이션에게 제공하는데, 키-값 쌍의 형식을 사용하여 상황을 모델링한다.

(2) 마크업 스키마 모델

모든 마크업 스키마(Markup Scheme) 모델링 기법은 속성과 내용을 갖는 마크업 태그로 구성된 계층적 데이터 구조이다. 특히, 마크업 태그의 내용은 항상 다른 마크업 태그에 의해 재귀적으로 정의될 수 있다. 이러한 종류의 상황 모델링 기법에서는 전형적으로 프로파일이 대표적이다. 프로파일들은 항상 XML과 같은 모든 마크업 언어의 수퍼클래스인 Standard Generic Markup Language(SGML)에서 유도된 직렬화에 기반한다.

2.3 논리 기반 모델

논리란 다른 표현이나 사실로부터 유도하여 결론적으로 어떤 표현이나 사실을 이끌어 낼 수 있는 조건들을 정의하는 것을 의미하며, 이러한 작업은 추측이나 추론이라고 알려진 처리 과정이다. 이러한 조건들을 규칙 집합으로 설명하기 위해서는 형식적인 시스템이 필요하다. 논리 기반의 상황 모델에서, 상황은 결과적으로 규칙, 사실, 표현들로써 정의된다. 초기의 논리 기반 상황 모델은 1993년 초기 Stanford 대학에서 McCarthy가 발표한 Formalizing Context[8]에서 찾아볼 수 있다.

2.4 온톨로지 기반 모델

온톨로지는 개념과 상관 관계를 기술하는 도구이다. 특히 온톨로지는 인간의 생활을 설명하고자 할 경우, 컴퓨터를 활용하여 그 일상 생활을 데이터 구조로 표현하는데 적합하다. 온톨로지 기반의 연구로는 Scale-ContextInformation(ASC)[Stra03]모델이 있다. 이 연구에서는 온톨로지를 사용함으로써 사실 및 하부 개념뿐 만 아니라 모델의 핵심 개념까지도 서술할 수 있는 단일화된 방법을 제공하는데, 유비쿼터스 컴퓨팅 시스템에서 상황 지식의 공유와 재사용을 지원한다.

III. 계층적 상황 온톨로지 관리 모델

상황 인식 서비스는 사용자, 디바이스, 서비스 등의 개체들간에 동일한 의미적 이해를 바탕으로 상황 정보를 교환할 수 있어야 한다. 이 장에서는 상황 인식 미들웨어에서 동작하는 상황 인식 서비스가 필요로 하는 상황 정보를 동적으로 변경될 수 있고, 미들웨어에 변경 내용을 전달함으로써 미들웨어로부터 해당 상황 정보를 전달받을 수 있도록 계층적 상황 온톨로지 관리 방법과 상황 모호성 문제를 해결할 수 있는 방법을 제안한다.

3.1 계층적 상황 온톨로지 구성

미들웨어상에서 관리되는 상황 정보는 미들웨어상의 모든 서비스에게 전달되는 도메인 상황정보와 서비스별로 별도로 정의되는 개별 상황 정보로 구성되며, 이들은 모두 온톨로지로 정의되도록 한다. 이때, 서비스가 미들웨어에 새롭게 배치되는 경우에는 미들웨어는 기존의 도메인 상황 정보와 서비스에서 별도로 정의된 개별 상황 정보를 통합 운용할 수 있어야 한다. 이를 위해서는 두 상황 정보들간의 관련성이 보장되어야 하며, 이를 해결하기 위해서 두 상황 정보 계층 사이에 공통 상황 정보 계층을 위치시킨다. 공통 상황 정보는 상황 인식 응용에서 필요로 하는 기본 요소를 사람(Person), 연산개체(CompEntity), 위치(Location), 환경(Environment), 행위(Activity)로 정의하고, 도메인 상황 정보와 개별 상황 정보는 공통 상황 정보를 상속받아서 설계하도록 한다. 이를 통해서 같은 부모 클래스로부터 상속받은 상황 정보끼리의 연관성을 제공함으로써 미들웨어에서 두 상황 정보를 통합하는데 이용되도록 한다. (그림 1)은 이 논문에서 설계한 계층적 상황 온톨로지를 표현한 것이다.

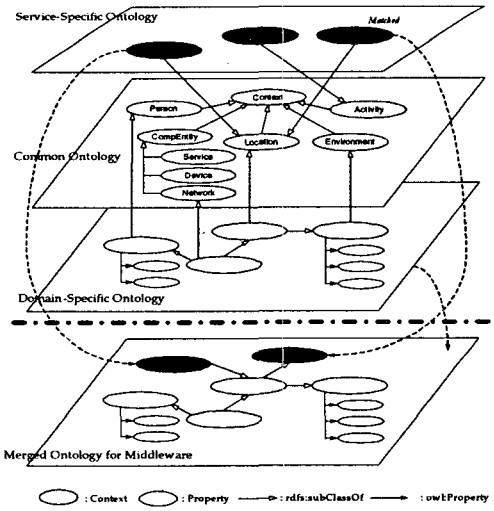

그림 1. 계층적 상황 온톨로지 구성
Fig 1. Hierarchical Context Ontology

모든 온톨로지는 Context 클래스에서 상속받아서 만들어진다. 공통 온톨로지에는 최상위 클래스인 Context 클래스에서 상속받은 Person, CompEntity, Location, Environment, Activity 클래스와 CompEntity에서 상속받은 서비스, 디바이스, 네트워크로 구성된다. 상황은 해당 상황을 설명하기 위한 다수의 속성을 포함할 수 있으며, 다른 상황을 프로퍼티로 포함할 수 있다. 예를 들어서, 홈네트워크 응용 온톨로지에서 Location 클래스로부터 상속받은 Room 상황과 Environment 클래스로부터 상속받은 Temperature 상황이 있는 경우 Temperature 상황은 Room 상황의 프로퍼티로 이용될 수 있다.

도메인 온톨로지 저작자는 이 공통 온톨로지를 상속받아서 미들웨어상의 모든 서비스에서 동일하게 해석되는 상황 정보를 정의한다. 각 상황 인식 서비스 개발자는 공통 온톨로지를 상속받아서 각 서비스에서 별도로 해석될 수 있는 개별 온톨로지를 정의한다. 이때, 상황인식 시스템은 두 온톨로지 계층 사이에 있는 상황 온톨로지를 중간 계층의 공통 온톨로지를 이용해서 연관성을 파악하고 이를 관리한다.

3.2 상황 충돌 해결 모델

앞에서 설명한 각각의 상황들은 시스템에 의해서 다양한 방법으로 파악될 수 있다. 일반적으로 상황인식에서 사용하는 방식은 사용자 및 주변 환경에 다양한 센서를 통해서 받은 원시 데이터를 통한 방법을 사용한다. 이렇게 얻어진 원시 데이터를 통해서 기본적인 상황 정보를 만들어내고, 다양한 추론을 통해서 보다 고차원적인 복합 상황을 유추하게 된다. 따라서, 여러 센서들로부터 전달받은 원시데이터에서 기본 상황을 만들어내기 위한 모델이 필요하며, 이 모델은 센서의 특성상 잘못 전달된 값이나 오류에 대한 적절한 필터링 방법을 이용해야 한다. 이를 위해서 상황 정보를 파악하고 필터링하기 위한 기본 모델을 제안한다.

3.2.1 상황

상황은 특정 시간에 센서로부터 수집된 여러 속성들이 일정 영역에 포함되는지에 따라서 결정된다. 예를 들어서, 온도 센서로부터 전달받은 사용자의 체온이 섭씨 36~36.5인 경우에는 사용자가 "건강한 상태"로 볼 수 있으나 그 이외의 값인 경우에는 "아픈 상태"로 정의할 수 있다. 따라서, 시스템은 센서로부터 받은 원시 데이터값을 통해서 상황을 판단할 수 있는 방법을 가져야 한다. 식 1는 이 논문에서 설계하는 미들웨어에 연결되어 있는 센서들로부터 특정시간(t)에 전달받은 속성들의 집합을 표현한 것이다.

$$S(t) = \sum_{i=1}^{n} A_i(t) \quad \text{(식 1)}$$

식 4에서 Ai(t)는 특정 시간 t에 속성 값을 나타낸 것이다. 상황 속성은 상황을 추론하는데 사용되어질 수 있는 요소들을 기술하는데 사용된다. 상황 속성은 가상 또는 물리적인 센서들과 연결되어 있다.

3.2.2 상황 공간

상황 공간은 특정 상황이 유추될 수 있는 속성들의 허용 가능한 값의 범위로 정의한다. 상황 공간은 미리 정의된 상황에 대응되기 위한 센서로부터 얻어지는 속성값들의 영역으로 표현된다. 식 2는 상황 공간을 표현한 것이다.

$$C_i = \sum_{i=1}^{n} S_i \qquad S_i = \{A \mid P(A)\} \quad \text{..... (식 2)}$$

P는 센서로부터의 속성값이 허용가능한지를 판단하는 Predicate 함수이며, A_i는 함수 P를 만족하는 요소들의 집합으로 정의된다. 상황(C)는 센서로부터 받은 속성값의 일정 범위(A)들의 합으로 표현된다. 센서에서 전달받은 값의 허용 범위인 A_i는 특정 값들의 집합으로 구성된다. C(t)를 통해서 속성값들의 허용 가능한 값들과 상황 정보를 연결시키고 있지만, 센서로부터 전달받은 속성값들을 통해서 바로 특정 상황정보로 맵핑되지는 못한다. 왜냐하면, 상황 A를 나타내는 C_A와 상황 B를 나타내는 C_B가 서로 겹쳐지는 부분이 존재할 수 있기 때문이다. (그림 2)는 두 상황이 겹쳐지는 상황을 표현한 것이다.

(그림 2)에서 S_A는 C_A에 포함되기 때문에 상황 A로 맵핑될 수 있다.

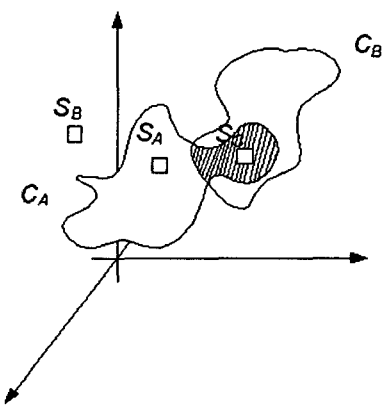

그림 2. 중첩되는 상황 공간
Fig 2. Duplicated Context Space

S_B는 C_A, C_B 어느곳에도 포함되지 않기 때문에 어떠한 상황에도 맵핑되지 못한다. 반면에 S_C는 C_A와 C_B의 중첩되는 부분에 위치하고 있다. 이런 경우에는 센서로부터 인식된 속성값들이 하나 이상의 상황 공간에 위치되기 때문에 어떤 상황 공간에 속하는지를 판단할 수 있는 필터링 과정이 필요하다.

이 경우 교차연산 연산자는 두 상황 공간간의 동일한 속성의 값을 갖는 공통의 영역을 포함하는 새로운 상황 공간을 만들어낸다. 두 상황공간의 교차연산에 포함되는 개체들은 다음의 식 5와 같이 정의할 수 있다. A는 조건함수 P를 만족하는 속성들의 집합이고, B는 조건 함수 Q를 만족하는 속성들의 집합일 때, 두 상황공간간의 교차연산에 포함되는 개체는 조건함수 P와 Q를 모두 만족하는 속성들로 표현된다.

$$A = \{V \mid P(V)\} \qquad B = \{V \mid Q(V)\}$$
$$a \in (A \cap B) \qquad iff \qquad P(a) \wedge Q(a) \quad \text{.... (식 3)}$$

센서로부터 전달받은 속성값들의 집합이 식 3과 같이 여러 상황 공간 내에 포함되는 경우에는 포함되는 상황공간과의 비교를 통해서 적절한 상황을 유추한다. 이를 위해서 이 논문에서는 속성값들과 상황공간과의 차이를 판단하기 위한 상황 공간 편차 함수를 정의한다. 이 상황 공간 편차 함수를 통해서 편차값이 작은 상황 공간으로 맵핑한다. 상황 공간 편차 함수는 식 4과 같이 정의한다.

$$Context - Space\ Derivation = \sqrt{\sum_{i=1}^{n} (A_s^i - A_r^i)^2}$$

$$A_s : Attibutes\ from\ Sensors$$

$$A_r : \frac{MaxValue(A_r) + MinValue(A_r)}{2} \quad \text{..... (식 4)}$$

예를 들어서, 사용자의 '달리는 상태'와 '걷는 상태'에서 각 상황공간에 위한 속성의 집합을 다음 〈표 1〉과 같이 정의할 수 있다. 센서로부터 받은 속성값인 Si = (165(H), 30(R), 36.61(B))인 경우, 이 Si는 어떤 상황 공간에도 완전히 포함되지 않는다. 시스템은 식 4를 이용해서 각 상황공간과의 편차를 계산한다. 계산결과 'Walking' 상황 공간편차가 'Running' 상황 공간편차보다 크기 때문에 현재 사용자의 상황을 'Running' 상황으로 유추해낸다.

표 1. 상황공간 속성 정의
Table 1. A Definition of Context Space Attributes

Context Space	Heart Rate	Respiration Rate	Body Temp
Walking	100-160	20-28	36.60-36.62
Running	150-200	26-40	36.61-36.63
Walking∩Running	160-160	26-28	36.61-36.62

Ⅳ. 미들웨어 설계 및 구현

상황 인식 미들웨어는 센서로부터 받은 데이터를 이용해서 상황 정보를 추론하고 이 정보를 원하는 상황 인식 서비스에 전달하는 역할을 수행한다. 상황 인식 미들웨어는 상황 인식 서비스의 개발 과정에서 상황 정보의 수집, 관리, 조합 및 상황 인식 서비스들에게 이들을 전달하는 역할을 담당한다.

상황 인식 미들웨어는 기본 상황을 생성하는 상황 생성기와 기본 상황을 복합 상황으로 추론하기 위해서 온톨로지 추론 엔진과 상호 작용을 위한 추론 엔진 인터페이스, 그리고 상황 인식 서비스와 공급/등록 방식으로 이벤트 전달을 위한 이벤트 브로커, 그리고 상황 저작자와 상황 인식 서비스로부터 전달받은 명령어를 해석하기 위한 번역기로 구성된다. 저작자는 3장에서 정의한 공통 상황 온톨로지를 이용해서 도메인에 필요한 상황 온톨로지를 정의한다. 정의된 온톨로지는 OWL 파일로 구성되며, 이를 번역기에 전달한다. 번역기는 상황 인식 서비스가 자신이 필요로 하는 상황 정보의 종류를 등록할 수 있는 메소드를 제공한다. 상황 저작자가 정의한 상황 온톨로지와 복합 상황을 추론하기 위한 규칙은 추론 엔진 인터페이스를 통해서 상황 추론 엔진에 전달되고, 상황 공간 정보는 기본 상황 생성기내의 데이터베이스에 저장된다. 기본 상황 생성기는 센서로부터 얻어진 데이터와 데이터베이스에 저장되어 있는 상황 공간 정보와 상황 충돌 처리기를 이용해서 적절한 기본 상황을 생성한다. 기본 상황 정보는 추론 엔진 인터페이스를 통해서 이벤트 전달자에게 전달되어서 상황 인식 서비스에 전달되고, 복합 상황 추론을 위해서 Fact로 변환된 뒤 추론 엔진에 전달된다. 추론엔진에 전달된 기본 상황은 상황 저작자가 정의한 규칙들과 온톨로지를 이용해서 일관성 검사와 복합 상황 추론 과정을 거치며, 이를 통해 생성된 복합 상황 정보는 다시 추론 엔진 인터페이스를 통해서 이벤트 전달자에게 전달되고, 최종적으로 해당 상황 정보를 필요로 하는 상황 인식 서비스에 전달된다. (그림 3)은 전체 상황 관리자의 시스템 구성도이다.

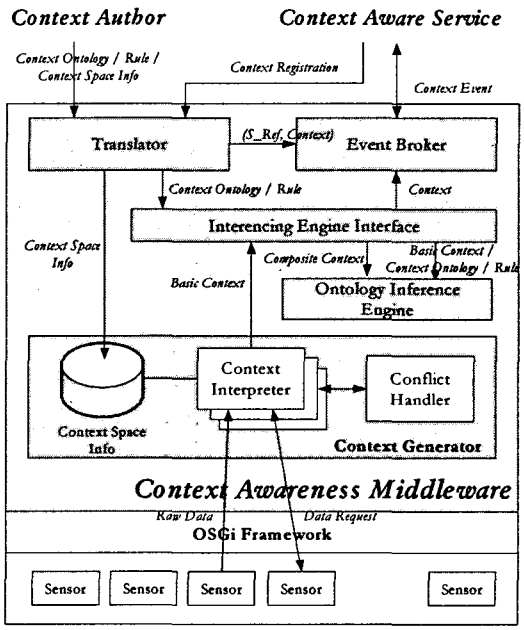

그림 3. 상황 인식 미들웨어 구성도
Fig 3. A Overview of Context Awareness Middleware

상황 인식 미들웨어는 OSGi 프레임워크 기반의 번들로 설계하였다. 따라서, 미들웨어는 OSGi 프레임워크의 BundleActivator 인터페이스를 상속받고, 온톨로지 추론을 위해서 Jena.jar 파일을 이용한다. 번역기는 온톨로지 저작자로부터 도메인 상황 정보와 규칙들을 전달받고, 상황 인식 서비스로부터는 개별 상황 정보를 파일 형태로 전달받을 수 있는 readFile() 메소드를 가지며, 내부적으로 OWL 파일을 파싱한 후, 온톨로지 추론 인터페이스와 번역기에 전달된다. 기본 상황 해석기는 센서로부터 얻은 정보와 상황 공간 정보를 비교해서 기본 상황을 생성하고, 만약, 상황공간간 충돌이 발생하는 경우에는 충돌 처리기의 resolveConflict() 를 이용해서 가장 적절한 상황 공간을 찾아낸다.

미들웨어 관리자가 OWL 파일 형태로 입력한 상황 온톨로지 정보와 상황 생성기에서 발생한 기본 상황 정보는 추론엔진 인터페이스를 통해서 Jena로 전달된다. 추론엔진 인터페이스는 온톨로지 추론 엔진인 Jena를 이용해서 복합 상

황 추론을 유추해 낸다. 이때, 기본 상황 생성기에서 발생한 기본 상황을 Fact의 형태로 추론엔진에 추가 및 삭제하기 위한 insertContext()와 removeContext()를 가지며, 미들웨어 관리자가 정의한 복합 상황 추론 규칙을 insertRule()과 removeRule()을 이용해서 관리한다. [리스트 1]은 추론엔진 인터페이스를 기술한 것이다.

[리스트 1] 추론엔진 인터페이스 구조

```
Interface InferencingEngineInterface {
FactID insertContext(Fact fact);     //Fact 삽입, 입력한 Fact의 ID 리턴
boolean removeContext(FactID factID; // FactID로 상황 삭제
uleID insertRule(Rule rule); //규칙 삽입, 입력한 규칙의 ID 리턴
boolean removeRule(RuleID ruleID); //RuleID로 규칙 삭제
boolean insertOntology(File ontologyFile);
//OWL파일로 도메인 온톨로지 추가
Boolean removeOntology(File ontologyFile); //온톨로지 삭제
}
```

추론엔진 인터페이스는 Fact, 규칙, 온톨로지를 추가 삭제하는 6개의 메소드로 구성된 인터페이스를 제공한다. 각 요소들은 추론엔진에 추가될 때 각각 ID를 부여받으며, 부여받은 ID는 해당 요소를 관리하기 위해서 사용된다.

기본 상황 생성기는 센서로부터 데이터를 전달받아서 이를 상황 공간 정보와 비교해서 적절한 기본 상황을 유추해 낸다. 이 상황 정보는 미들웨어에서 관리되기 위해서 상황 ID를 부여하고, 추론엔진 인터페이스를 통해서 Fact로 변환된 후 다시 추론 엔진에 전달된다. 추론엔진에서는 Fact와 규칙을 이용해서 온톨로지 추론과 사용자 규칙 기반 추론을 통해서 상황 정보의 일관성 체킹과 복합 상황을 생성한다. 이렇게 생성된 상황 정보는 이벤트 브로커에 전달되고, 이벤트 브로커는 상황 정보를 등록한 상황 인식 서비스에게 전달한다.

V. 실험 및 평가

이 장에서는 이 논문에서 설계한 상황 인식 미들웨어의 기능과 성능을 평가하기 위해서 스마트 홈네트워크 서비스들을 구현하여 실험을 진행하였다. 홈네트워크 서비스 환경에 사용될 서버는 IBM eServcer X206, 2.8GHz, 512MB RAM이며, 윈도우 서버 2003 환경에서 OSGi 프레임워크인 Knopflerfish 1.3.3을 사용하고, HP의 시멘틱 웹 툴킷인 Jena2를 이용하였다.

5.1 스마트 홈 네트워크 구현

실험 시나리오는 사용자가 집안으로 들어오면서 동적으로 발생하는 상황 정보들을 생성하고 이에 따라서 상황 인식 서비스에 해당 상황 정보가 적절하게 전달되는지를 보인다. (그림 4)는 실험에서 구현한 홈네트워크 환경을 도식화한 것이다. 사용자가 1번 위치에서 2번 위치로 이동하는 경우 RFDI 리더로부터 사용자의 존재를 확인하고 이에 적절한 서비스를 구동하도록 설계하였다.

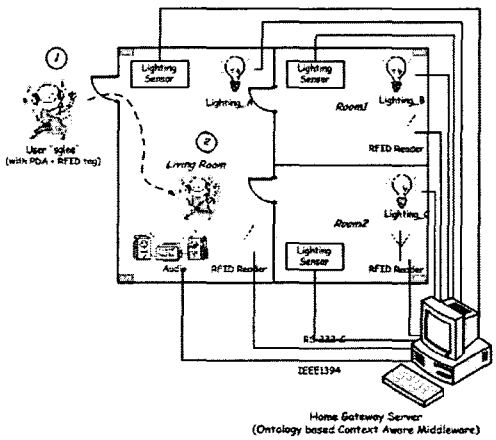

그림 4. 스마트 홈네트워크
Fig 4. Smart HomeNetwork

조명 제어 서비스는 사용자 위치에 있는 장소의 조명이 어두우면 자동적으로 사용자와 가장 가까운 조명을 켜주는 상황 인식 서비스이다. 조명 제어 서비스는 조도 센서를 통해서 얻어진 정보를 이용해서 각 장소의 밝기 정도를 상황으로 인식한다. 가정 내 밝기를 파악하기 위해서 Room1, Room2, LivingRoom에 조도 센서를 설치한다. 조도 센서는 빛의 세기를 감지해서 10Bits로 표현하기 때문에 0~1023까지의 값을 가지며, 센서로부터 얻어진 값이 512 이하인 경우에는 "Dim" 상황으로 되고 512 이상인 경우에는 "Bright" 상황으로 유추한다. 조명 제어 서비스는 '사용자가 있는 위치가 어둡다'와 '조명이 켜져야 한다"라는 개별 상황을 [리스트 2]와 같이 정의한다.

[리스트 2] 개별 상황 정의

```
(?LightSensor locateIn ?place), (?LightSensorValue
bigger 512) -> (?place lightLevel "Bright")
(?LightSensor locateIn ?place), (?LightSensorValue
smaller 512) -> (?place lightLevel "Dim")
(?user locatedIn ?place), (?place lightLevel "Dim") ->
(?place needed "lighting")
```

(그림 4)에서 사용자 "glee" 1번 위치에서 2번 위치로 이동하는 경우, RFID 리더 3번은 사용자가 부착한 RFID 태그로부터 사용자의 ID를 얻는다. RFID 리더는 이 정보를 RFID 관리 서비스에 전달한다. RFID 관리 서비스는 해당 정보를 파싱해서 사용자 ID값은 미들웨어에 전달한다. 기본 상황 생성기는 ("glee" ocatedIn "ivingRoom") 정보를 생성한다. 생성된 기본 상황 정보는 온톨로지 추론엔진에 전달되며, 조명 제어 상황 서비스에 개별적으로 전송된다.

집안의 각 방과 거실에는 조도 센서를 이용해서 주기적으로 방과 거실의 조도를 측정하고, 그 값을 미들웨어에 전송한다. 미들웨어는 각 장소에서 측정된 조도 센서의 값을 리스트 2에 따라서 512보다 큰 경우에는 "밝다"라는 상황을 생성하고, 512보다 적은 경우에는 "어둡다"라는 상황을 생성한다.

센서로부터 얻어진 값이 각각 ("Room1" lightingValue 284), ("Room2" lightingValue 653), ("LivingRoom" lightingValue 327)이었으며, 이 정보를 통해서 ("Room1" lightLevel "Dim"), ("Room2" lightLevel "Bright"), ("LivingRoom" lightLevel "Dim")의 기본 상황을 생성하였다. 이 상황은 추론 엔진에 전달되며, 추론엔진은 이미 가지고 있는 지식중 ("sglee" locatedIn "LivingRoom")과 (?user locatedIn ?place), (?place lightLevel "Dim") -> (?place needed "ighting"을 통해서 ("ivingRoomg" needed "lighting") 복합 상황 정보를 추론해 내었다. 이 추론된 복합상황은 이 상황을 등록한 조명 제어 서비스에 이벤트 브로커를 통해서 전달된다.

5.2 성능 평가

이 논문에서 제안한 미들웨어는 상황 인식 서비스가 동적으로 상황 온톨로지를 등록할 수 있기 때문에 기존 연구에 비해서 온톨로지를 메모리에 로딩하는 시간 이외에 온톨로지 통합의 오버헤드가 존재한다. (그림 5)에서 미들웨어에서 관리하는 상황 정보의 수에 따라서 상황 정보를 메모리에 로딩하는 시간과 도메인 상황과 개별 상황을 통합하는 시간을 측정하였다.

이 오버헤드를 측정한 측정한 결과 (그림 6)과 같이 온톨로지 로딩 시간에 비해서 통합 시간이 큰 부담이 아니며, 도메인 상황 온톨로지의 숫자와 개발 상황 온톨로지의 숫자가 증가폭에 비해서 통합 시간의 증가폭이 크지 않음을 알 수 있다. 따라서, 상황 정보 통합에 소요되는 시간으로 인해서 이 연구에서 제안한 미들웨어는 실시간 서비스의 플랫폼으로는 적합하지 않으나, 일반적인 응용 서비스에는 유용하게 이용될 수 있을것으로 보인다.

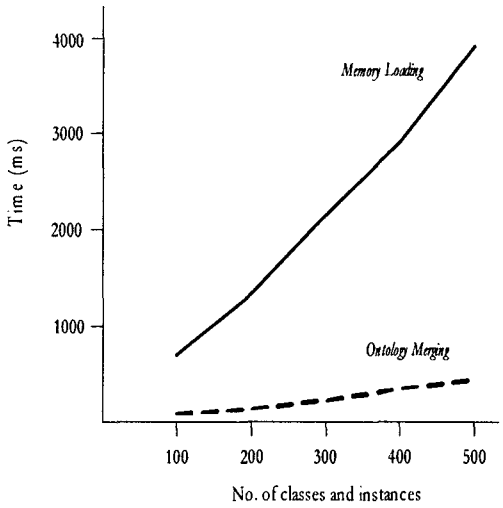

그림 5. 상황 온톨로지 통합 오버헤드
Fig 5. A Merging Overhead of Context Ontology

이 오버헤드를 측정한 측정한 결과 (그림 6)과 같이 온톨로지 로딩 시간에 비해서 통합 시간이 큰 부담이 아니며, 도메인 상황 온톨로지의 숫자와 개발 상황 온톨로지의 숫자가 증가폭에 비해서 통합 시간의 증가폭이 크지 않음을 알 수 있다. 따라서, 상황 정보 통합에 소요되는 시간으로 인해서 이 연구에서 제안한 미들웨어는 실시간 서비스의 플랫폼으로는 적합하지 않으나, 일반적인 응용 서비스에는 유용하게 이용될 수 있을것으로 보인다.

Ⅵ. 결론

이 논문에서는 유비쿼터스 컴퓨팅 환경에서 온톨로지 기반 상황 인식 서비스 관리 모델을 제시하였고, 제시한 모델을 이용해서 상황 인식 서비스 미들웨어를 설계 및 구현하였다. 설계한 미들웨어는 상황 인식 서비스가 동적으로 필요로 하는 상황 온톨로지를 미들웨어에 등록할 수 있도록 함으로써 상황 인식 서비스별로 개별화된 상황 정의를 지원하고 기존 연구에서 상황 인식 서비스 설계시 상황 정보를 설계해야 하는 단점을 해결하였다. 또한, 설계된 미들웨어는 센서와 가전등의 물리적 기기와 연결하기 위해서 OSGi 프레임워크 환경에서 구현하였다.마지막으로 이 논문에서 설계/구현한 미들웨어의 타당성을 입증하기 위해서 홈네트워크용 상황 인식 서비스를 구현하였다. 실험을 통해서 미들웨어의 기능과 성능을 평가하였으며, 제안한 계층적 상황 온톨로지 관리 모델의 오버헤드로 인해서 하드 실시간(Hard Realtime) 유비쿼터스 컴퓨팅 환경에는 적합하지 않으나, 일반적인 유비쿼터스 컴퓨팅 환경에는 별 무리없이 적용될 수 있음을 확인하였다. 이 논문에서 제시한 온톨로지 기반 상황 인식 미들웨어는 홈네트워크, 텔리매틱스, 스마트 오피스 등의 다양한 유비쿼터스 환경에서 서비스 게이트웨이에 탑재되어서 상황 인식 서비스 운용 환경으로 적용될 수 있을 것으로 보인다.

참고문헌

[1] A. K. Dey, "Providing Architectural Support for Building Context-Aware Applications," PhD Thesis, Georgia Institute of Technology, 2000.

[2] N. Q. Hung, S. Y. Lee, and L. X. Hung, "A Middleware Framework for Context Acquisition in Ubiquitous Computing Systems", Proceedings of the Second International Conference on Computer Applications, 2004.

[3] M. Weiser, "The Computer for the Twenty-First Century," Scientific American, pp. 94-101, 1991.

[4] T. Strang, "Service Interoperability in Ubiquitous Computing Environments," PhD Thesis, Ludwig-Maximilians University, 2003.

[5] T. Buchholz, M. Krause, C. Linnhoff-Popien, and M. Schiffers, "CoCo: Dynamic Composition of Context Information," Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, pp. 335-343, 2004.

[6] A. Arsanjani, F. Curbera, and N. Mukhi, "anners Externalize Semantics for On-demand Composition of Context-aware Services," roceedings of the IEEE International Conference on Web Services, pp. 583-590, 2004.

[7] A. K. Dey, "Supporting the Construction of Context-Aware Applications," Dagstuhl seminar on Ubiquitous Computing, 2001.

[8] B. N. Schilit and M. M. Theimer, "Disseminating Active Map Information to Mobile Hosts," IEEE Network, Vol. 8, pp. 22-32, 1994.

[9] McCarthy and S. Buvac, "Formalizing Context (Expanded Notes)," Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language (Menlo Park, California, 1997), S. Buvac and L. Iwanska, Eds., American Association for Artificial Intelligence, pp. 99-135, 1997.

[10] T. Strang, "Service Interoperability in Ubiquitous Computing Environments," PhD Thesis, Ludwig-Maximilians University, 2003.

[11] 이승근, "유비쿼터스 환경을 위한 서비스 게이트웨이 간이 서비스 이동 관리 시스템 개발," 한국컴퓨터정보학회논문지, 2005. 12.

[12] 김효남, 박용, "유비쿼터스 컴퓨팅 환경에서 상황인식 미들웨어 설계," 한국컴퓨터정보학회논문지, 2005. 11.

저 자 소 개

이 승 근

1996년 2월 인하대학교
전자계산공학과
1998년 2월 인하대학교
전자계산공학과 공학석사
2006년 2월 인하대학교
컴퓨터정보공학과 공학박사
2000년 ~ 2005년 (주)하이켐텍
기술연구소 책임연구원
2006년 ~ 현재 : 인하대학교
컴퓨터공학부 강의전담교수
〈관심분야〉 유비쿼터스 컴퓨팅,
상황인식, 홈네트워크,

김 영 민

1988년 2월 인하대학교
전자계산학과
1990년 2월 인하대학교
전자계산학과 이학석사
1998년 8월 충북대학교 대학원
전자계산학과(박사수료)
1992년 ~ 현재 : 부산경상대학
의료정보과 조교수
〈관심분야〉 유비쿼터스 컴퓨팅,
상황인식, 의료정보화