상황정보의 확장성을 지원하는 상황정보 기반 컴포넌트 모델

(Context Driven Component Model Supporting Scalability of Context)

윤 회 진 † 최 병 주 ††
(Hoijin Yoon) (Byoungju Choi)

요 약 상황정보기반 컴포넌트 모델은 상황인식 어플리케이션이 상황정보에 민감한 부분과 그렇지 않은 부분으로 구성된다는 아이디어에서 출발하여, 상황정보에 민감한 부분을 각 상황정보의 모습과 수행 조건에 따라 분할하고 각각을 소프트웨어 컴포넌트로 구성하는 모델이다. 본 모델은 상황인식 컴포넌트들을 조합하는 방식으로 어플리케이션을 구현함으로써, 상황인식 어플리케이션의 상황정보에 따른 확장성을 지원한다. 더불어 어플리케이션 소스코드 자체에 내재되어져 있는 상황정보를 어플리케이션 로직과는 분리시킴으로써, 변화하는 상황정보에 효율적으로 적응하도록 한다. 본 모델이 갖는 특성을 보이기 위하여, 대표적인 상황인식 어플리케이션의 하나인 Call-forwarding 어플리케이션 구현에 제안한 모델을 적용하고, 이를 통해 실제 상황정보 확장에 대한 지원과 상황정보의 분리를 어떻게 해결하는지를 분석한다.

키워드 : 상황인식, 컴포넌트 기반 개발, 유비쿼터스 컴퓨팅

Abstract Since Context Driven Component model is based on the idea that a context-aware application consists of the components that are context sensitive and the components that do not depend on the context, it divides the context sensitive part into components according to which context information they are related to. The model supports the scalability of context information by building an application through composing Context Driven Components. Furthermore, it solves the embeddedness of context information inside the application logic. To show the contributions of the model, this paper applies it to Call-forwarding application, and analyses how the model supports the scalability and the embeddedness.

Key words : Context-awareness, Component-based Development, Ubiquitous Computing

1. 서 론

유비쿼터스 컴퓨팅의 대두로 다양한 소프트웨어에 대한 요구사항이 변하고 있다. 그 가운데 기술적 차원에서, 대규모의 유비쿼터스 소프트웨어가 동적으로 변화하는 환경에 적응하는 능력을 고려한 소프트웨어 설계와 구현에 대한 기술적인 문제들이 있다[1]. 즉, 유비쿼터스 컴퓨팅에서의 소프트웨어는 유기적으로 성장해야 한다[2]. 기존의 컴퓨팅 환경과 달리, 유비쿼터스 컴퓨팅이 적용되는 실세계는 동적으로 변화한다. 결국 유비쿼터스 컴퓨팅을 지원하는 소프트웨어는 동적으로 변화는 상황에 자연스럽게 적응하면서 수행되어져야 하며, 이 특성을 가능하게 하는 개발 기술이 요구되는 현실이다. 상황정보에 따라 소프트웨어의 행위를 변화시킬 능력이 요구되며, 이 능력을 상황인식(Context Awareness)라고 부른다. 유비쿼터스 컴퓨팅에서 상황정보는 더욱 자주 변화하게 되므로, 상황인식은 유비쿼터스 컴퓨팅의 중요한 조건이 된다.

따라서 유비쿼터스 소프트웨어에 대한 접근은 상황인식 소프트웨어로 시작할 수 있다. 그러나 기존의 상황인식 소프트웨어들은 처리해야 할 상황정보들이 상황인식 소프트웨어 구현 코드 자체에 내재되어져 있다. 보통 규칙기반시스템으로 상황인식소프트웨어를 작성하며, 규칙 내부에 상황정보가 포함되어져 있으며, 해당 행위들이 하나의 소프트웨어에 함께 표현되어져 있다. 본 논문은

· 이 논문은 2006년도 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(No.KRF-2006-531-D00027)
† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
hjyoon@ewha.ac.kr
†† 종신회원 : 이화여자대학교 컴퓨터학과 교수
bjchoi@ewha.ac.kr
논문접수 : 2006년 7월 21일
심사완료 : 2006년 12월 28일

상황인식 소프트웨어 개발을 컴포넌트 단위로 구성하는 모델을 제안하여, 상황정보의 변화에 따른 행위 변경, 행위 추가, 처리해야 할 상황정보 추가 등을 컴포넌트 단위로 수행할 수 있도록 한다. 이를 통해, 유비쿼터스 컴퓨팅 환경에 맞는 대규모 상황인식 소프트웨어로의 합리적인 확장 기술을 제공할 수 있으며, 상황인식에 따른 행위 변화를 전체 소프트웨어를 대상으로 수행하지 않고 해당 컴포넌트만을 대상으로 수행 가능하도록 한다. 본 논문이 제안하는 컴포넌트는 상황정보에 따라 구분되는 소프트웨어 컴포넌트이므로, "상황정보 기반 컴포넌트 (Context Driven Component : CDC)"로 명명한다. 본 논문은 상황인식 소프트웨어 요구사항을 분석하여 CDC를 설계하는 모델을 제안한다.

다음의 세 가지 직면 문제들이 본 모델의 개발 근거가 된다.

첫째, 유비쿼터스 컴퓨팅에서 상황인식 소프트웨어는 잦은 상황정보 변화에 적응해야 한다. 상황정보는 소프트웨어가 수행되는 동안에 동적으로 변화한다. 위치인식 소프트웨어의 경우, 변화하는 위치 정보에 따라 소프트웨어는 다른 행위를 보여줘야 하며[3], 위치정보는 실시간으로 동적으로 변화할 것이다. 변화하는 상황정보에 맞추는 상황인식 소프트웨어의 적응성은 외부의 간섭 없이 자연스럽게 이루어짐으로써, "*Calm computing*"을 실현할 수 있다[4]. 이것이 유비쿼터스 컴퓨팅에서의 상황인식 소프트웨어가 직면한 문제이며, 본 논문에서 제안하는 모델은 이 문제를 컴포넌트 기반 소프트웨어 개발 개념으로 해결하고 있다.

둘째, 유비쿼터스 컴퓨팅에서는 상황정보의 규모가 점점 커지고 있다. 과거에는 상황인식 연구가 위치인식으로 국한되는 경우가 많았으나, 유비쿼터스 컴퓨팅에서의 상황인식은 위치인식 그 이상의 의미를 갖는다[5]. 이는 현재의 위치뿐 아니라, 주변 객체들과의 전반적인 관계에 대한 인식을 의미한다. 따라서 상황정보의 규모가 커지는 것을 염두에 두어야 하며, 상황정보 규모에 대하여 유연하게 대처할 수 있는 구조로 소프트웨어를 개발하는 모델이 요구된다. 본 논문의 개발 모델은 상황정보에 따라 소프트웨어 컴포넌트를 구성하고 이를 통합하는 구조의 개발 모델로서, 상황정보 규모 변화에 유연하게 대처할 수 있다.

셋째, 소프트웨어 코드로부터 상황정보를 분리시키는 것이 유비쿼터스 컴퓨팅에서 운용되는 소프트웨어 개발에 중요하다. 일반적으로 상황인식 소프트웨어는 상황정보를 소스코드 내에 포함시킨다. 그러나 특히 유비쿼터스 컴퓨팅에서는 상황정보 변화가 자주 발생하고 소프트웨어는 그에 따라 동적으로 변화해야 하므로, 이러한 포함관계는 소프트웨어가 상황정보 변화를 받아들이는 것을 어렵게 한다. 상황정보가 변화할 때마다 전체 소프트웨어를 탐색하여 적합한 행위를 찾아 수행해야 한다. 상황정보가 분리된다면, 해당 상황정보에 민감한 행위 부분을 찾아 해당 행위를 수행하도록 할 수 있는 효율성을 갖을 수 있다. 본 논문은 상황정보를 따로 관리하고 상황정보마다의 행위를 소프트웨어 컴포넌트로 구현함으로써, 상황정보 변화에 좀 더 효율적으로 대처할 수 있도록 한다. 본 논문은 위의 세 가지 문제 상황을 기반으로 추출한 세 가지 요구사항 - 상황정보의 적응성, 상황정보의 확장성, 그리고 상황정보 분리 - 을 해결한다.

2장에서는 유비쿼터스 컴퓨팅을 고려한 상황인식 소프트웨어 개발에 관한 기존 연구들을 소개하고, 본 논문의 관점과의 차이점을 기술한다. 3장에서는 CDC 설계 방안과 전체 개발 구조를 설명하며, 앞서 언급한 세 가지 요구사항을 어떻게 해결하고 있는지를 언급한다. 4장에서는 유비쿼터스 소프트웨어의 하나인 액티브 배지의 Call-forwarding 어플리케이션을 본 모델로 구현하는 예를 보인다. 5장에서는 기존 연구들과 본 논문의 제안 모델을 비교 분석하고 6장 결론에서 본 논문이 갖는 의의와 결론을 기술한다.

2. 관련 연구

상황정보를 다루는 연구는 크게 두 가지로 분류되어진다. 하나는 상황정보를 데이타로서 수집·추상화하는 영역과 다른 하나는 어플리케이션 입장에서 상황정보를 어떻게 받아들이고 그에 적응할 것인가 하는 영역이다. 본 논문의 제안은 두 번째 영역에 해당하는 것으로서, 본 장에서는 기존의 유비쿼터스 환경을 염두에 두고 진행되었던 상황인식 어플리케이션 개발 모델에 대한 설명과 각 모델의 관점에서 보는 본 연구의 위치를 비교한다.

2.1 컨텍스트 툴킷

컨텍스트 툴킷은 개발에 있어서, 상황정보 수집 및 추상화 부분과 상황인식 어플리케이션의 분리를 목적으로, 센서에서부터 최종 서비스까지 상황정보 추상화 정도에 따라 각각의 수준을 독립적인 컴포넌트 단위로 구현하는 모델이다. 전체 컴포넌트들은 XML파일 형태의 공통적인 통신 매체를 사용하고, 이들은 HTTP로 전송되어지며, 각 컴포넌트들의 조합으로 최종의 상황인식 어플리케이션이 개발되는 구조이다. 이들 컴포넌트들은 각각 독립적으로 수행되고 있으므로, 컴포넌트 기반 개발이 갖는 재사용성의 효과도 얻을 수 있다.

상황인식 어플리케이션을 구성하는 컴포넌트들을 다음의 5가지로 제안하고 있다[6].

• *Widget* : 실제 센서로부터 들어오는 상황정보를 어플리케이션에게 접근시키는 역할을 하는 소프트웨어 컴

포넌트로서, 실제 센서와의 복잡성을 해결해주고, 센서로부터 들어오는 데이타를 어플리케이션이 원하는 모습으로 추상화시킨다. 어플리케이션의 관점에서 *Widget*은 상황정보를 캡슐화하고, 상황정보에 접근할 수 있는 오퍼레이션을 제공한다.

• *Interpreter* : 상황정보의 추상화 수준을 올리는 역할을 하는 소프트웨어 컴포넌트로서, 예를 들어, 위치정보는 위도와 경도로 표현하거나, 더 높은 수준으로 도시 또는 거리 이름으로 표현할 수 있다. 나아가 받아들인 상황정보로부터 새로운 상황정보를 유추할 수도 있다.

• *Aggregator* : 논리적으로 연결되어지는 상황정보들을 모아서 저장소에 넣어놓고, 사용되도록 한다.

• *Application* : 어플리케이션 행위를 수행하는 소프트웨어 컴포넌트로서, 어떤 일련의 조건들이 이루어지면 어플리케이션이 해당 행위를 수행한다.

• *Discoverer* : 프레임워크에 어떤 컴포넌트들이 존재하는가에 대한 리스트를 유지관리하는 소프트웨어 컴포넌트로서, *widget, interpreter, aggregator, application*을 지속적으로 관찰한다. 이들 컴포넌트들 가운데 어느 하나가 시작되면, discoverer는 그 존재와 능력을 감지하고, 어떻게 그와 연결해야 하는지를 알려준다.

컨텍스트 툴킷은 다섯 개의 컴포넌트들이 그림 1에 표현한 대로, 센서로부터 들어오는 상황정보를 주요 관점으로 하는 Widget 컴포넌트로부터 상위 서비스를 주요 관점으로 하는 Application 컴포넌트까지 계층 개념으로 구성되어져 있다. 이 가운데 Aggregator를 통해 관리되는 수준의 상황정보들의 갱신을 Application 컴포넌트에서 어떻게 적용할 것인지에 대한 부분이 본 논문에서 제안하는 방법이다.

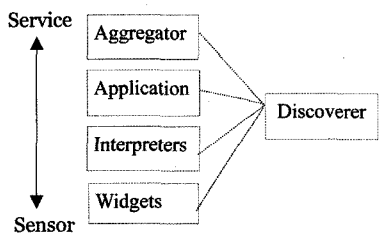


그림 1 컨텍스트 툴킷의 컴포넌트들

2.2 Gravity

Gravity[7]는 서비스를 통한 상호작용을 하는 컴포넌트 모델이다. 목적은 실행시간에 서비스 존재의 다양성에 자동으로 적응하는 능력을 갖는 컴포넌트 기반 어플리케이션 구축을 지원하는 것이다. 동적으로 변화되는 서비스에 적응하는 부분으로서, 응용부분을 유비쿼터스 컴퓨팅에서의 상황정보 변화에 적응하는 것으로 내세우고 있어, 본 논문의 제안 기술과 일맥상통하는 연구이다. Gravity는 실행시간에 컴포넌트들을 조립하는 방법으로서 서비스 개념을 사용하였다. Gravity는 컴포넌트 기반 어플리케이션 개발에 초점을 맞추고 있으며, 본 논문은 상황인식 어플리케이션 개발에 초점을 맞추고 있다. 이 두가지는 모두 컴포넌트의 개념과 서비스의 개념을 사용하고 있다. 그러나 서로 다른 의미로 사용되고 있다. Gravity는 컴포넌트를 서비스 기반 개념이 지원하는 몇가지 기술 내용 - 웹서비스의 WSDL이나 SOAP 기술 내용 - 을 컴포넌트에 덮어씌워서 컴포넌트 서비스를 구현하는 모델이고, 본 논문의 모델은 유비쿼터스의 상황인식 어플리케이션을 수행하는 서비스 구현을 위해 요구되는 상황정보에 따른 컴포넌트를 설계하고 이들을 통해 하나의 서비스를 구성하게 함으로써 상황정보 변화에 어플리케이션이 동적으로 적응할 수 있도록 한다. 그림 2(a)는 Gravity의 서비스와 컴포넌트의 개념적 위치관계이고, 그림 2(b)는 본 논문의 모델이 갖는 서비스와 컴포넌트의 관계이다.

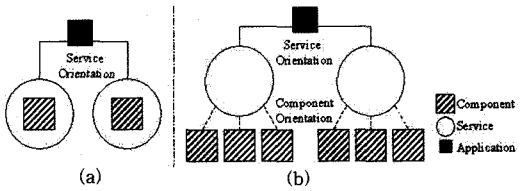


그림 2 Gravity와의 비교 (컴포넌트와 서비스의 개념적 위치)

2.3 Multifacet

Multifacet[8] 접근법은 본 논문과 동일하게 상황인식 어플리케이션이 변하는 상황정보에 적응하는 방안을 제시한다. 이는 중앙에서 어플리케이션 행위들의 수행 순서를 수행 조건과 우선순위등에 기반하여 관리한다. 어플리케이션은 상황정보에 민감한 컴포넌트와 상황정보와 상관없는 컴포넌트로 구성되어진다는 아이디어에서 출발한 방법이다. 본 논문의 방법의 아이디어도 이와 유사하여, 어플리케이션을 상황정보기반 컴포넌트들로 구성하고 그에 해당하지 않는 부분을 별도의 컴포넌트로 구성한다. Multifacet 접근법은 상황정보에 대한 적응을 위해 변경되어져야 하는 부분을 컴포넌트의 일부분인 facet이라는 단위로 구현하고, 컴포넌트들마다 facet들을 갖고 있어서 그 부분의 변경을 통한 어플리케이션을 facet만을 통해 수행한다.

이 방법의 경우, 적응을 위해서는 컴포넌트의 일부분으로 구현된 facet에 수정을 가하기 위하여 컴포넌트 자체를 수정해야 하는 로드가 생기는 반면, 우리의 접근법은

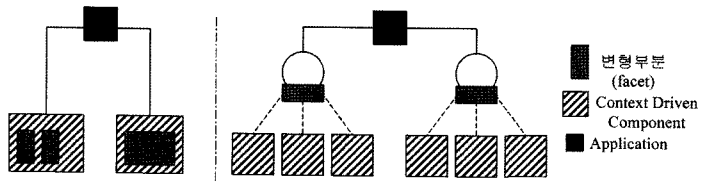


그림 3 Multifacet과의 비교 (컴포넌트와 서비스의 개념적 위치)

컴포넌트들은 그 자체로 완전한 기능을 수행하며, 상황정보 변화를 반영하는 곳은 중간에 환경설정파일들을 통해 이루어진다. 그림 3은 이 두 방법에서의 컴포넌트와 기타 적응을 위한 변경 부분의 개념적 위치관계를 나타낸다.

3. 상황정보 기반 컴포넌트 모델

기존의 상황인식 소프트웨어들은 여러 개의 If-then 구조를 갖는 규칙기반시스템으로 작성되었다. 이는 앞서 기술한대로 상황정보들과 그에 관련된 기능들이 소프트웨어 내부에 녹아들어가 있는 구조이다. 본 논문은 상황정보에 대한 확장성과 적응성을 확보하기 위하여, 상황정보 기반 컴포넌트로 모듈화하여 상황인식 소프트웨어를 구현하는 모델을 제안한다.

그림 4는 본 논문에 제안하는 상황정보 기반 컴포넌트 모델을 표현한 것이다. 빗금친 상자가 상황정보기반 컴포넌트를 나타내고, 그 위의 노란색 동그라미는 상황정보기반 컴포넌트들을 조합해 놓은 서비스는 의미한다. 하나의 상황정보기반 소프트웨어는 유비쿼터스 환경에서 하나의 서비스로 존재할 수 있으며, 이는 향후 서비스 기반 아키텍처(Service Oriented Architecture : SOA)로 엮을 경우, 다른 서비스와 함께 좀 더 큰 서비스로서 동작할 수 있다.

본 논문에서 제안하는 개발 모델의 핵심 기술은 상황정보를 기반으로 설계한 소프트웨어 컴포넌트에 있다. 상황정보기반 컴포넌트는 상황정보들의 특정 조건을 만족할 때 발생하는 행위의 구현이다. 즉 특정 조건에서 수행해야 하는 행위를 하나의 CDC로 구현함으로써, 상황정보 변화에 따른 적응성과 상황정보 규모에 대한 확장성을 지원할 수 있다.

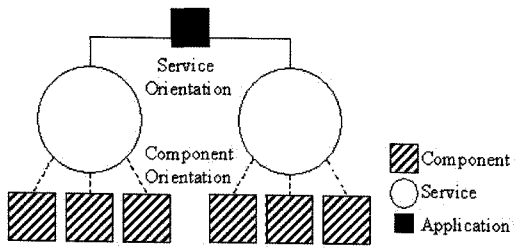


그림 4 컴포넌트, 서비스, 어플리케이션의 위치관계

3.1 상황정보와 수행 조건

상황인식 소프트웨어는 일반적으로 규칙기반 시스템으로 구현된다. 규칙 기반 시스템들은 조건과 그 조건일 때 수행해야 할 행위를 규칙으로서 사용한다. 이때 행위는 내부 함수의 형태로 구현이 되거나 아주 간단한 경우에는 규칙내부에 간략하게 기술될 수 있다. 따라서 시스템 내부에 다양한 상황정보들과 그에 대한 행위들이 내재되어져 있다. 본 논문에서는 이러한 규칙을 상황정보에 맞추어 표현하기 위하여, 다음의 내용으로 조건을 기술한다.

condition = (context type, context information)

*context type*은 상황정보의 종류를 표현한다. 예를 들어 위치정보를 상황정보로 사용한다면, 상황정보 종류는 '위치'가 될 수 있다. 유비쿼터스에서의 상황정보의 종류는 행위(Activity), 신원(Identity), 위치(Location), 그리고 시간(Time)으로 나뉘어진다[9]. 위의 *context information*은 *context type*에 적용되는 실제 감지된 정보값을 의미한다. 이는 센서로부터 감지된 상황정보가 서버로 전달되고 서버에서 추상화를 수행하여 어플리케이션에서 요구하는 정제된 형태로 상황인식 소프트웨어에 전달된다. 본 논문은 서버 수준에서의 상황정보가 아닌 정제되고 추상화가 이루어진 상황정보를 의미한다. 예를 들면, 센서에서 서버에 전달하는 상황정보가 위도와 경도로 표현되는 정보였다면, 이 상황정보가 서버에서 추상화가 이루어진 모습은 해당 위치에 대하여 상황인식 소프트웨어가 요구하는 형태의 정보로 변형된다. '부엌'이나 '거실' 등이 그 예가 될 수 있다.

상황인식 소프트웨어에서 상황정보들을 인식한다는 것은 현재의 상황정보를 받아들이고 그에 적응하여 해당 행위를 수행해야 하는 것이다[10]. 상황정보가 일정 '조건'을 만족하면 그에 해당하는 행위를 수행한다. 즉 조건은 상황정보의 영향을 받으며, 행위는 조건의 영향을 받는다. 본 논문에서는 특정 조건 Ci를 만족할 때 수행해야 하는 함수를 Fi를 F(Ci)로 표현한다. 함수를 결정하는 가능한 조건들의 조합은 표 1과 같다.

표 1 조건들의 조합 관계

Relation	Description
$F(C_x)$	조건 Cx를 만족할 때 함수를 수행하는 경우
$F(C_x \wedge C_y)$	조건 Cx와 Cy를 동시에 만족할 때 함수를 수행하는 경우
$F(C_x) \neq F(C_x)$	조건 Cx를 만족할 때 수행해야 하는 함수가 두개 이상인 경우
$F(C_x \vee C_y)$	Cx또는 Cy를 만족할 때 함수를 수행하는 경우

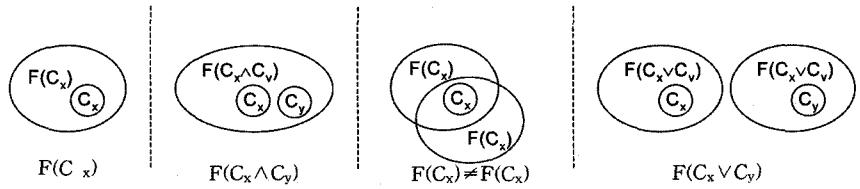


그림 4 조건 조합 관계에 따른 기능과 조건들의 연관관계

조건과 함수를 표현하는 다이어그램으로 위의 각 경우를 나타내면 그림 5와 같다.

예를 들어, TOM이 거실에 나타날 때 수행해야 하는 행위를 표현해보면, C_x를 (Identity, TOM), C_y를 (Location, 거실)라고 할 때 $F(C_x \wedge C_y)$로 나타낼 수 있다.

3.2 조건에 따른 행위와 상황정보기반 컴포넌트

앞의 3.2.1의 과정을 통하여 상황정보로부터 조건이 만들어지고, 각 조건들의 조합에 따라 수행해야 하는 행위가 결정된다. 상황정보와 조건에 의해 결정된 행위, 즉 함수 F는 관련되는 상황정보에 따라 하나의 소프트웨어 컴포넌트로 구현된다. 이때 어떤 상황정보와 어떤 함수가 관련이 되어있느냐에 따라, F_{simple}, F_{single}, $F_{multiple}$로 나눌 수 있다.

F_{simple}은 하나의 상황정보에 영향을 받는 함수로서, F(Cx)의 Cx가 하나의 상황정보에 따른 조건인 경우이다. 그림에서 F1이 F_{simple}이다. F_{single}은 그림 6의 F2가 해당되는 경우로서, 복수의 상황정보의 영향을 받는 함수이다. 이때 관련되는 상황정보들은 하나의 *context type*에 속한다. 서로 다른 *context type*에 속하는 상황정보들의 영향을 받는 함수는 $F_{multiple}$로 정의하고, 그림의 F3이 $F_{multiple}$에 속한다.

상황정보들 사이의 coupling을 최소화 하는 방향으로 상황정보기반 컴포넌트를 구현하여야 소프트웨어 컴포넌트로서 갖는 장점을 살릴 수 있다. 그러나 상황정보시스템에서 $F_{multiple}$의 경우가 흔히 발생한다. 예를 들어, "TOM이 거실에 들어오면, TOM이 선호하는 프로그램이 재생된다."라는 시나리오에서 'TOM'은 Identity에 속하고 '거실'은 Location에 속한다. 이 경우에 수행되는 F, "TOM이 선호하는 프로그램을 재생한다",는 $F_{multiple}$에 속한다.

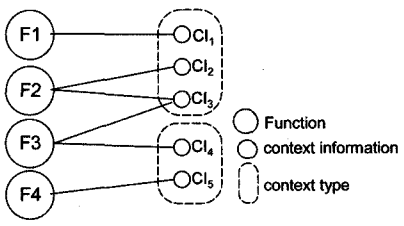
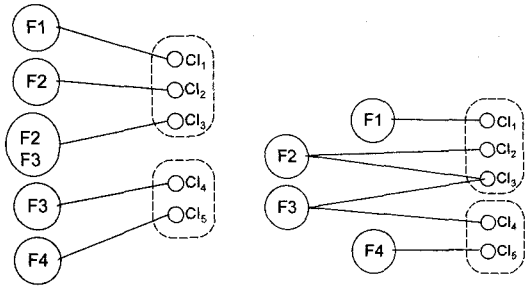


그림 6 기능과 상황정보와의 연관관계

본 논문에서는 소프트웨어 컴포넌트를 상황정보를 기준으로 나누어, 상황정보 변화에 따른 적응성과 확장성을 목표로 한다. F_{single}과 $F_{multiple}$의 경우, 해당되는 *context information*들 사이의 관계가 AND 또는 OR의 관계를 갖는다. OR의 경우는 두 가지 이상의 *context information*에 영향을 받기는 하지만, 서로 동시에 영향을 받지는 않는 경우로서, 각 *context information*에 해당하는 컴포넌트로 구현할 수 있다. AND의 경우는 앞서 'TOM'과 '거실'의 예처럼 두 가지 정보가 동시에 고려되는 경우로서, 상황정보기반 컴포넌트들 사이의 결합력을 유발시키는 관계이다. 이러한 함수는 독립적인 소프트웨어 컴포넌트로 구성하되, *context information* 사이의 결합력이 없는 그 외의 소프트웨어 컴포넌트와는 다른 레벨로 관리하여, 우리가 지원하는 상황정보기반 컴포넌트의 의미를 유지한다. 따라서 일단 F_{single}과 $F_{multiple}$의 경우 OR의 관계를 전제하여 다음과 같은 방법으로 F_{simple}로 변형시킨다. 그림 7은 위 그림에서 F_{single}과 $F_{multiple}$이 OR인 경우와 AND인 경우, 상황정보기반 컴포넌트를 나누어 놓은 그림이다. (b)의 F2와 F3의 경우는 두 가지 이상의 상황정보에 동시에 영향을 받는 경우로서, *context information* 하나에 따른 컴포넌트로 구현할 수 없다. 그 자체를 하나의 컴포넌트 단위로 구현하여야 하며, 이러한 관계에 놓이는 *context information* 또는 *context type*은 연관성이 높은 것들로서, 구현상 분리할 경우 더 많은 오버헤드를 초래할 수 있다.



(a) OR관계일 경우 (b) AND관계일 경우

그림 7 $F_{multiple}$ 분석 및 연관관계 재구성

3.3 구현 모델

상황인식을 위한 조건과 그에 따른 함수를 정의하고, 각 함수가 관련되어 있는 상황정보에 따라 상황정보기반 컴포넌트로 분류하는 방법에 대한 설명을 앞장에서 하였다. 구성된 컴포넌트는 전체 상황인식 소프트웨어의 어떤 위치에서 어떻게 동작하는지에 대한 전체 구조는 그림 8과 같다. 상황정보기반 컴포넌트들은 상황정보관련 서비스에 의해 사용되는 형태이며, 서비스와 CDC사이에는 이들 관계에 대한 메타데이타인 CIConfiguration 파일이 존재한다. 그림의 점선 상자 하나의 모습은 SOA의 기본 구현 기술인 웹서비스의 구조와 유사하다. 향후 이 모델은 SOA 개념을 포함하도록 확장하기 위한 구조이다. 따라서 CIConfiguration 또한 웹서비스의 WSDL과 SOAP 메시지와 같이 XML문법을 따르도록 구현하였다.

앞의 그림에서 보았듯이 상황정보 서비스 - 그림 9에서의 context service - 는 다른 상황정보 서비스들과 연결되어서 좀 더 큰 규모의 서비스를 구성할 수 있다.

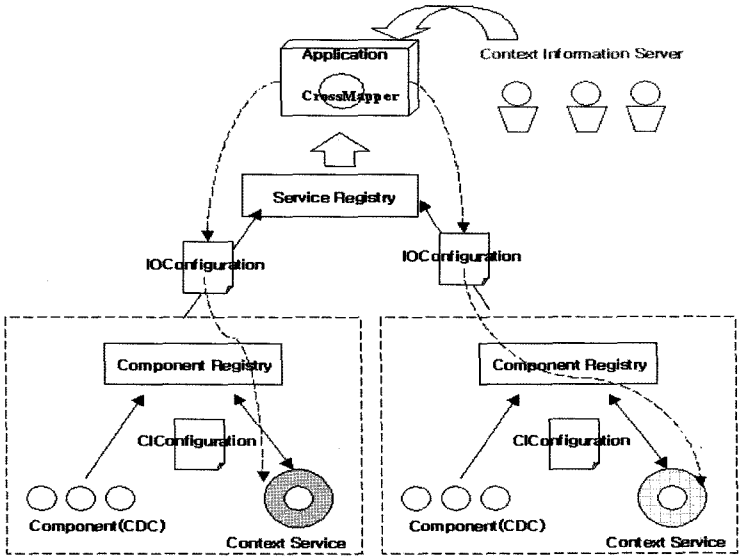


그림 8 상황정보기반 컴포넌트 (CDC)를 이용한 어플리케이션 수행 모델

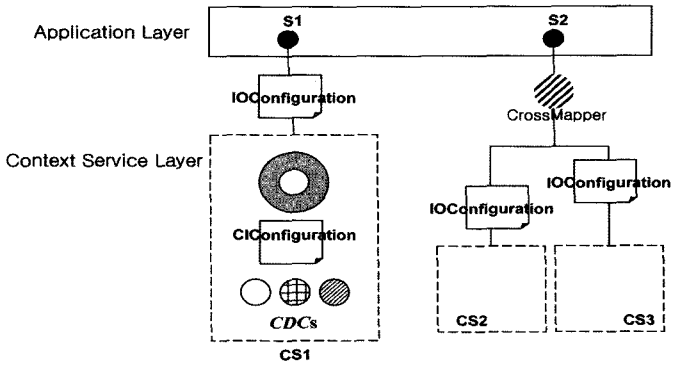


그림 9 CDC를 이용한 어플리케이션 구현 구조

상황정보서비스들 사이의 연결을 위한 메타데이타를 IOConfiguration에서 관리하도록 하고, 서비스들은 독립적으로 존재하면 실행시에만 연결되어 동작하는 개념을 갖는다. 이렇게 함으로써, 상황정보에 따른 함수의 추가나 삭제가 용이하게 된다. 이는 컴포넌트 기반 소프트웨어 개발이 갖는 장점을 상황인식 소프트웨어 개발에 적용한 것이다. 서비스들을 서로 엮어주는 역할에 포함되는 CrossMapper는 SOA 지원 도구인 *Fiorano*의 IOMapper와 유사한 기능을 한다.

상황인식 소프트웨어는 그림의 S1 또는 S2의 형태로 어플리케이션 층에 등록될 수 있다. S1은 하나의 상황정보서비스로 이루어져 있으며, S2는 복수의 상황정보서비스로 이루어져 있다. $F_{multiple}$가운데 AND로 연결되어진 상황정보들은 하나의 상황정보 서비스 내에 구현되어진다. 위 그림에 따르면, 상황정보는 어플리케이션의 로직으로 부터 분리되어 관리된다. 상황정보는 CIConfiguration에 존재하고, 그에 따른 행위들은 CDCs로 구현되어진다. 이 둘이 서로 연동하여 서비스가 운영된다. 이로 인해 두 가지 효과가 발생한다. 서비스 내부의 코드 변화없이 상황정보를 갱신할 수 있고, 두개 이상의 서비스의 조립을 내부 소스코드 수정없이 가능하게 한다. 이 두 가지 특징을 각각 CIConfiguration과 IOConfiguration에서 지원한다. CIConfiguration은 상황정보들과 관련된 CDC들의 리스트를 관리하고, IOConfiguration은 다른 CS에게 공개되어야 하는 부분을 관리한다. 이는 향후 SOA 기반으로 확장할 때, Coordination 서비스 수준으로 구현될 것이다.

4. 상황정보 기반 컴포넌트 모델 적용 예

본 논문이 제안한 상황정보 기반 컴포넌트 모델이 어떻게 실제 구현에 적용되는지와 어떻게 확장성을 지원하는지를 보이기 위하여, 본 장에서는 액티브 배지 시스템의 Call-forwarding[11]에 본 모델을 적용하여 CDC를 추출하고 이들을 조합하여 어플리케이션을 구현한다.

4.1 적용 도메인

본 장은 액티브 배지 시스템의 Call-forwarding 어플리케이션에 본 논문에서 제안하는 모델을 적용한다. 액티브 배지의 Call-forwarding 어플리케이션은 상황인식을 나타내는 최초의 어플리케이션일 것이다. 사용자들 모두는 액티브 배지를 몸에 부착하고 있어야 한다. 액티브 배지는 적외선 송신기로서 각 사람에 해당하는 ID를 센서에 송신한다. 사용자가 움직이면, 데이타베이스는 동적으로 사용자의 현재 위치에 대한 정보를 갱신한다. Call-forwarding 어플리케이션은 전화가 어떤 사람에게 오면, 교환원은 데이타베이스를 사용하여 현재 그 사람이 위치하는 곳에서 가장 가까운 전화기로 신호를 보내

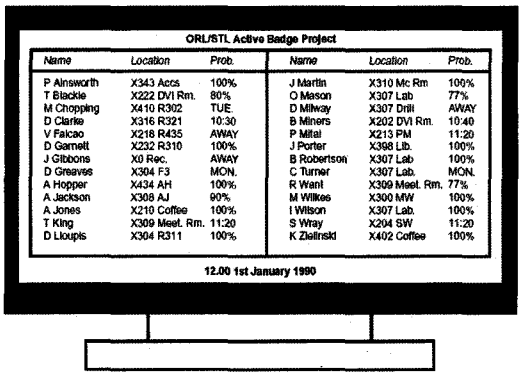


그림 10 Call-forwarding 어플리케이션 서버 화면샷

통화를 전달해준다. 서버는 액티브 배지 센서들을 뒤져서 현재의 위치 정보를 지속적으로 유지해 주어야 한다.

위의 그림은 서버의 화면을 보여준다. 센서로부터 얻어진 정보는 서버로 넘겨져, 그림 10과 같은 결과를 만들어 화면에 보여준다. 첫 번째 필드는 이름으로서, 현재 소속된 멤버들의 이름을 나열하고 있다. 두 번째 필드는 현재 해당 멤버의 액티브 배지가 감지되는 곳의 위치를 나타내고 있다. 세 번째 필드는 어떤 사람이 현재 위치로 잡힌 곳에서 찾아질 확률을 표시한다. 이 값이 100%이하이면 그 멤버가 다른 곳으로 다니고 있다는 것을 의미한다. 만일 5분 동안 멤버의 액티브 배지가 센서에 감지되지 않으면 마지막 감지된 시간을 세 번째 필드에 넣고, 두 번째 필드에 마지막 감지된 장소를 넣는다. 센서에서 감지되지 않은 시간이 24시간이 경과되었으면, 마지막 감지된 날(요일)을 세 번째 필드에 넣는다. 또한 일주일이상 위치정보 갱신이 없다면 AWAY로 표시한다. 센서가 감지하는 매 순간마다 - 배지에서 15초마다 센서로 신호를 보내고 있다 - 제대로 감지되는 경우에만 세 번째 필드에 100%가 채워진다.

4.2 CDC 설계 및 구현

액티브 배지의 Call-forwarding 어플리케이션의 기본 기능만을 수행하는 프로토타입을 본 논문에서 제안한 상황정보 기반 컴포넌트들로 구현하였다. 단, 서버 전송 정보 가운데 세번째 필드는 모두 100%로 간주하고, 모든 사람이 감지된 위치에 항상 있음을 전제하였다. 우선 상황정보들로 이루어지는 조건들의 종류를 살펴보고, 각 조건들과 요구되는 기능들의 관계를 분석한 후, CDC를 구성한다. 조건은 6개로 분석될 수 있으며, 다음과 같다.

C1=(Name, TOM), C2=(Name, Jane), C3=(Name, Kate),

C4=(Location, Area A), C5=(Location, Area B), C6=(Location, Area C)

Call-forwarding의 기능은 "전화를 받는다", "전화를

해당사람의 위치로 전달한다" 두 가지로 나뉘어지며, 다시 두 번째 기능을 세분하면, "전화를 Area A로 전달한다.", "전화를 Area B로 전달한다", "전화를 Area C로 전달한다"로 나뉘어진다. 따라서 총 4개의 기능을 갖는다. "전화를 받는다."는 상황정보와 관련없는 기능으로서, CDC를 구성하는데 제외되며 F0로 표시한다. 나머지 3개의 기능은 각각 F1, F2, F3로 표시하며, 각각은 다음의 조건 조합에 의해 수행된다.

$F1=F((C1 \wedge C4) \vee (C2 \wedge C4) \vee (C3 \wedge C4))$

$F2=F((C1 \wedge C5) \vee (C2 \wedge C5) \vee (C3 \wedge C5))$

$F3=F((C1 \wedge C6) \vee (C2 \wedge C6) \vee (C3 \wedge C6))$

F1, F2, F3는 $F_{multiple}$의 관계를 갖으며, 내부적으로 각 기능들은 OR의 관계로 연결되는 AND 관계들을 갖고 있다. 따라서 각 기능들은 OR로 연결되어있는 각 AND 조합에 대한 기능으로 복제 될 수 있으며, 더 세분화되서 AND 조합 자체를 갖는 기능은 $F_{multiple}$의 수준으로 남겨진다. 이를 기반으로 CDC를 F1, F2, F3를 각각 표현하는 소프트웨어 컴포넌트로 구현하고, 이 각각은 $F_{multiple}$의 형태로서 그림 11과 같은 구조를 갖는다.

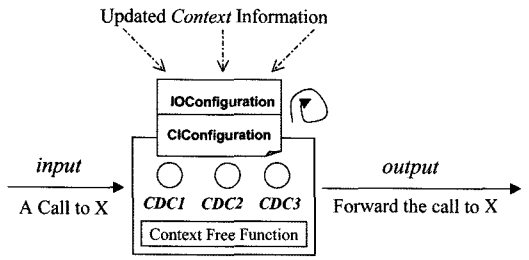


그림 11 Call-forwarding의 입출력과 상황정보 관계

앞서 언급했던 "전화를 받는다."와 같은 상황정보과 관계없는 기능과 함께 구현하는 CDC1~CDC3가 어플리케이션내에 존재하고, 입력과는 다른 방향으로 센서에 의해 실시간으로 감지되는 상황정보들이 적용된다. 갱신되는 상황정보들은 CDC 자체에 영향을 주지 않으며, 단지 IOConfiguration과 CIConfiguration을 갱신한다. 어플리케이션은 수행 중에 지속적으로 Configuration 파일들을 참조하므로, 변화된 상황정보를 출력에 반영할 수 있다. 그림 12는 사용되었던 CIConfiguraion.xml파일의 어느 한 순간의 내용이다.

본 논문에서는 CDC1, CDC2, CDC3를 .NET 환경에서 수행되는 dll파일로 구현하였다. 구현 환경은 Window XP에서 Visual Studio .NET 2003을 사용하였다. 임의로 전화가 걸려온 것을 가정하여, 수행한 화면이다. Configuration 파일을 참조하여 현재 어떤 dll파일이 구동되어져야 하는지를 찾고, 해당 dll 파일을 수행하였다. 각 dll파일은 실제 수행해야 하는 일에 대한 설명을

```
<Context name="Location">
  <Info>
    <Content>Area A</Content>
    <Component onoff="off">CDC1.dll</Component>
  </Info>
  <Info>
    <Content>Area B</Content>
    <Component onoff="off">CDC2.dll</Component>
  </Info>
  <Info>
    <Content>Area C</Content>
    <Component onoff="on">CDC3.dll</Component>
  </Info>
</Context>
```

그림 12 CIConfiguration 샘플 파일

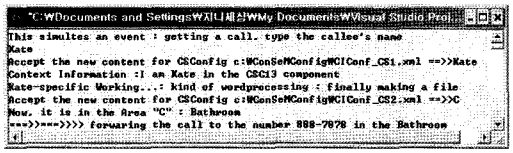


그림 13 프로토타입 수행 샘플 화면

출력하는 것으로 구현하였다. 이를 수행한 결과 화면이 그림 13과 같다.

4.3 상황정보의 확장성 지원

유비쿼터스 컴퓨팅 환경이 확대되면서, 상황정보 규모 또한 확장되고 있다. 본 논문에서 제안한 접근법은 확장되는 상황정보에 효율적으로 대처하여 이미 가지고 있는 어플리케이션을 갱신하도록 한다. 상황정보를 기반으로 컴포넌트들을 작성하여 운용되는 어플리케이션을 구현함으로써, 새로운 상황정보를 추가하거나, 상황정보가 삭제되어질 때, 추가되는 상황정보에 따른 새로운 기능을 정의하여 컴포넌트로 독립적으로 구현하고, 이를 IOConfig와 CIConfig라는 환경설정파일에 등록시키는 방법으로 새로운 상황정보를 받아들이도록 어플리케이션을 수정할 수 있다. 삭제의 경우, 해당되는 컴포넌트에서 관련되는 조건을 삭제하거나, 조건의 전체가 삭제되는 상황정보와 관련이 된다면, 해당 컴포넌트 자체를 제거할 수 있다. 이는 기타 다른 컴포넌트에 영향을 최소화하는 방향으로 수정을 진행 할 수 있도록 한다.

그림 12와 같이 앞의 Call-Forwarding의 최소 버전

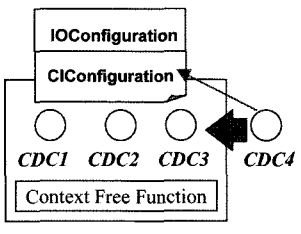


그림 14 컴포넌트 추가및 등록

에서 대상이 되는 Location의 상황정보에 Area D가 추가되어야 한다면, C7=(Location, Area D)로 하는 F((C1∧C7)∨(C2∧C7)∨(C3∧C7))가 하나의 CDC4로 구현되고, 이를 CIConfiguration과 IOConfiguration에 등록시킨다. 이렇게 함으로써, 어플리케이션 코드 내부를 모두 수정하지 않고, 새로운 상황정보를 인식하는 어플리케이션으로 변형시킬 수 있다. 이는 컴포넌트기반 개발이 갖는 장점에서 비롯되었으며, 상황정보를 구현 컴포넌트 내부가 아닌 환경설정파일(Configuration)로 따로 분리시킴으로써, 상황정보 변화에 대한 내용을 환경설정파일에 수정하는 것으로 상황정보에 대한 변화를 어플리케이션에 반영할 수 있다.

5. 분 석

유비쿼터스 컴퓨팅은 실시간으로 변하는 다양한 상황정보들을 인식해야 한다. 이는 상황인식 어플리케이션의 특징이다[10]. 이러한 상황인식 어플리케이션은 실시간으로 변화하는 상황정보에 적응해야하는 '적응성'이 요구된다. 기존의 상황인식 어플리케이션들은 모든 가능한 상황정보와 그에 대한 행위를 규칙기반 시스템 등의 기존 구현 모델로 작성하여 상황정보와 어플리케이션 로직의 소스코드가 뒤섞여 있는 모습이었다. '상황정보와 소스코드의 분리'는 상황정보 변화를 소스코드 없이 적응할 수 있도록 하는 기본 조건[12]으로서 유비쿼터스 컴퓨팅의 구현 모델의 특성의 하나로 꼽을 수 있다. 나아가 기존의 상황정보에서 새로운 상황정보를 추가로 인식해야 하는 경우, 기존의 상황정보의 어플리케이션을 유지하면서 새로운 상황정보를 인식하는 코드를 추가할 수 있는 '확장성' 또한 유비쿼터스 컴퓨팅 환경의 확대와 함께 요구되고 있다. 또한 최근 구현 모델의 효율성 측면에서 많은 연구들이 '컴포넌트 기반 기술'과 '서비스 기반 기술'의 개념을 사용하고 있다. 컴포넌트 기반 기술은 적응성을 용이하게 하기 위한 모델로서 고려되고 있으며, 서비스 기반 기술은 향후 서비스 기반 아키텍처와 연동 가능한 모델로의 가능성을 부여하기 위한 노력이다.

지금까지 기술한 적응성, 확장성, 상황정보 분리, 컴포넌트 기반 기술, 서비스 기반 기술이 앞서 2장에서 언급한 기존 연구들과 본 논문에서 제안하는 상황정보기반 컴포넌트 모델에서 어느정도 표현되고 있는지를 표 2로 나타내었다. ×는 지원하지 않음을 나타내고, ○는 지원함을 나타낸다.

컴포넌트 기반 기술은 컨텍스트 툴킷과 Multifacet 그리고 본 논문의 모델에서 보여진다. 컨텍스트 툴킷은 상황인식 어플리케이션의 작성을 컴포넌트 단위로 하는 구현 모델이다. 각 컴포넌트는 센서가 상황정보를 받아들이는 부분, 즉 센서와의 결합도가 높은 부분에서부터 정제된 상황정보를 서비스 차원에서 사용하는 부분, 즉 하드웨어 또는 센서로부터 감지된 상황정보를 추상화하여 사용하는 상위 수준의 어플리케이션 부분까지를 구성하는 각 단계의 컴포넌트 모델을 제안한다. 이는 상황정보가 하단에서 정리되고, 그를 사용하는 서비스 로직을 상단에서 구현하도록 계층적으로 구성되어진다. Multifacet은 상황정보에 민감한 컴포넌트와 민감하지 않은 컴포넌트로 전체 어플리케이션을 구성하고, 상황정보에 적응하기 위하여 컴포넌트내에 facet이라는 부분을 변형의 대상으로 한다. 본 논문의 상황정보기반 컴포넌트는 상황정보의 조건들을 분석하고 연관된 조건들 마다 수행해야 하는 행위를 컴포넌트로 구현한다.

서비스 기반 기술은 향후 상황인식 어플리케이션을 서비스 기반 아키텍처의 서비스로 구현하기 위한 기본 기술이다. Gravity는 컴포넌트의 조립 부분을 서비스화하여 SOA의 약한 결합력 특성을 통한 실시간 컴포넌트 조립을 제안하였다. 본 논문에서 제안한 상황정보기반 컴포넌트도 향후 상황인식 어플리케이션을 SOA의 하나의 서비스로 구성하기 위한 그림 4의 기본 구조를 갖는다.

적응성은 상황인식의 기본 요건으로서, 변화하는 상황정보에 맞는 행위를 실시간에 변경하여 수행하는 특성이다. Gravity는 주변에 존재하는 컴포넌트들을 실시간 조립하는 모델로서, Gravity의 컴포넌트가 상황정보의 조건에 따른 행위 구현이라면, 변경하는 상황정보에 맞는 행위를 구현한 컴포넌트들을 실시간으로 조립하여 현재 상황정보에 적응할 수 있다. Multifacet도 facet들을 수정하는 방법으로 적응성을 표현할 수 있다. 상황정보기반 컴포넌트의 경우, 변경되는 상황정보들을 환경설정파일에 실시간으로 적용함으로써, 제공되는 컴포넌트들 가운데 현재 상황정보와 연관된 조건에 따른 행위를

표 2 기존 연구들과의 비교 결과표

	컨텍스트 툴킷	Gravity	Multifacet	상황정보기반 컴포넌트
컴포넌트 기반 기술	○	×	○	○
서비스 기반 기술	×	○	×	×
적응성	×	○	○	○
확장성	×	×	×	○
상황정보 분리	○	×	×	○

구현한 컴포넌트를 실행한다. 이렇게 함으로써, 적응성을 지원하고 있다.

확장성은 어플리케이션이 고려해야 하는 상황정보가 추가되거나 수정될 때, 기존의 상황인식 어플리케이션을 최대한 재사용하여, 확장된 상황정보에 맞추어 수행하는 어플리케이션을 구성할 수 있는 특성을 의미한다. 본 논문에서 제안하는 상황정보기반 컴포넌트는, 확장되는 상황정보와 연관된 조건에서 수행해야 하는 행위를 독립된 컴포넌트로 구현하고, 그 컴포넌트에 대한 정보를 환경설정파일에 등록시킴으로써, 추가되는 상황정보의 인식을 가능하게 한다. 이는 어플리케이션을 상황정보를 기준으로 나눠지는 컴포넌트로 구성하였고, 상황정보와 관련되는 컴포넌트의 정보를 어플리케이션 소스코드와는 별도로 환경설정파일에 둠으로써, 기존 코드의 수정을 최소화하면서 상황정보를 확장할 수 있는 확장성을 지원한다.

상황정보 분리 문제는 이미 여러 연구에서 대두되었던 이슈이다. 상황정보와 관련된 내용을 어플리케이션 소스코드와 분리시킴으로서, 적응성 확장성을 효율적으로 지원할 수 있다는 의미에서이다. 컨텍스트 툴킷 개발의 기본 아이디어도 이와 유사하다. 센서로부터 들어오는 상황정보에서부터 시작하여 최종의 어플리케이션 수행 로직까지 단계별로 상황정보의 추상화 정도에 따라 계층적으로 컴포넌트들을 구성하여 상황인식 어플리케이션을 개발하도록 하였다. 이는 결과적으로 상황정보에 밀착되는 컴포넌트와 어플리케이션 서비스에 밀착되는 컴포넌트들로 최종 어플리케이션을 구성한다. 본 논문의 컴포넌트도 상황정보를 환경설정파일로 분리해서 구성하고, 실제 수행 코드는 컴포넌트내에 존재하도록 하여, 상황정보를 분리시켰다. 이렇게 함으로써, 효과적인 적응성을 지원하며 나아가 확장성을 위하여 컴포넌트 내부 코드 변경을 최소화하였다.

6. 결 론

본 논문에서 상황정보기반 컴포넌트를 설계하고 이를 이용하여 상황인식 어플리케이션을 개발하는 구현모델을 제안하였다. 이 모델이 갖는 이점은 1장에서 언급했던 현재 상황인식에 대한 세 가지 요구사항과 연관된다. 첫째, 상황정보 변화를 지속적으로 받아들이고 그에 적응할 수 있다. 이러한 상황정보 변화에 대한 적응성은 환경설정파일을 지속적으로 갱신시키면서, 그에 따라 순간순간 요구되는 기능을 갖는 상황정보기반 컴포넌트들을 찾아서 호출함으로써 해결하였다. 둘째, 상황정보 규모 변화에 대한 효율적인 대처를 들 수 있다. 이는 앞의 4.3장에서 설명한 것처럼 컴포넌트기반 개발이 갖는 장점에서 비롯되는 특징이다. 추가되거나 변경되는 상황정보를 기반으로 소프트웨어 컴포넌트를 구현하고 그들을 상위의 메타모델로서의 CIConfig와 IOConfig의 환경설정파일에 수정사항을 접수함으로써, 내부 다른 코드를 수정하지 않고 컴포넌트 수준의 수정만으로 상황정보에 대한 확장성을 보장할 수 있다. 셋째, 기존 연구들에서 지적되었던 어플리케이션 코드 내에 상황정보가 내재되는 문제를 해결하여 상황정보와 어플리케이션 로직을 분리하였다. 이렇게 함으로써, 상황정보에 대한 변화를 어플리케이션 로직 내에 직접 적용하지 않아도 되는 효과를 볼 수 있으며, 이로 인해 앞의 첫 번째와 두 번째 효과를 얻을 수 있게 된다.

본 연구는 유비쿼터스 컴퓨팅 패러다임에 부합하는 모델임을 [13]에서 보였고, 현재 모델 구성과 그에 대한 프로토타입 구현을 통해 제안한 모델의 수행 가능성을 보였으며, 기존 연구들과의 비교 분석을 통하여 본 논문에서 제안하는 구현 모델의 이점을 기술하였다. 향후 더욱 다양한 상황인식 어플리케이션에 적용하는 구현을 통해 본 모델의 효율성을 실험을 통해 분석하고, 이를 지원하는 분석 및 설계 도구를 구축할 계획이다. 또한 4장에서 언급한대로 SOA의 서비스로서의 상황인식 어플리케이션을 구축하기 위하여 SOA의 기본 요건[14]인 자율성(Autonomous), 재사용성(reusability), 무상태성(statelessness), 발견가능성(discoverability)를 상황인식 어플리케이션에 부여하는 연구를 진행하고 있다. 이를 통해 본 구현 모델에 서비스 기반 기술의 특성을 부여할 수 있으며, 향후 SOA와의 연동을 가능하게 한다.

참 고 문 헌

[1] Kalle Lyytinen and Youngjin Yoo, "Issues and Challenges in Ubiquitous Computing," Communications of the ACM, Vo.45, No.12, pp.63-65, Dec. 2002.

[2] F. Brooks. Mythical Man-Month. Addison Wesley, 20th anniversary edition, 1995.

[3] Brown, P.J., Bovey, J. D. and Chen, X., "Context-Aware Applications: From the Laboratory to the Marketplace," IEEE Personal Communications, Vo.4, No.5, pp.58-64, Oct. 1997.

[4] Tim Kinberg and Armando Fox, "System Software for Ubiquitous Computing," Pervasive Computing, January-March, pp.70-81, 2002.

[5] A. Schmidt, M. Beigl, H. W. Gellersen, "There is more to context than location," Proceedings on the Intl. Workshop on Interactive Applications of Mobile Computing (IMC98), Rostock, Germany, Nov. 1998.

[6] Anind K. Dey and Gregory D. Abowd, "The Context Toolkit: Aiding the Development of Context-Aware Applications," in Proceedings of

Workshop on Software Engineering for Wearable and Pervasive Computing, 2000.
[7] Humberto Cervantes and Richard S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model," in Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 2004.
[8] Anca Rarau, Kalman Pusztai, and Ioan Salomie, "MultiFacet Item Based Context-Aware Applications," InternationalJournal of Computing & Information Sciences, Vol. 3, No. 2, 2005, pp.10-18.
[9] Anind K. Dey, Providing Architecrural Support for Buildeing Context-Aware Applications, Ph.D Thesis, Gerogia Institute of Technology, 2000.
[10] Gregory D. Abowd, "Software Engineering Issues for Ubiquitous Computing," Proceedings of ICSE '99, pp.75-84, May 1999, LA, CA, USA.
[11] Roy Want, Andy Hopper, Veronica Falcao and Jonathan Gibbons, "The Active Badge Location System," ACM Transactions on Information Systems 10(1):00.91-102, Jan. 1992.
[12] Karen Henricksen and Jadwiga Indulska, "Developing context-aware pervasive computing applications : Model and approach," Pervasive and Mobile Computing, Vol.2, No.1, pp.37-64.
[13] Hoijin Yoon, Byoungju Choi, "The Context Driven Component Supporting the Context Adaptation and the Content Extension," Journal of Information Science and Engineering, Accepted.
[14] Thomas Erl, *Service Oriented Architecture*, Prentice Hall, 2005.

윤 희 진

1993년 이화여자대학교 전자계산학과 이학사. 1998년 이화여자대학교 컴퓨터학과 이학석사. 2004년 이화여자대학교 컴퓨터학과 이학박사. 2005년~현재 이화여자대학교 컴퓨터학과 전임강사. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 상황인식어플리케이션, 유비쿼터스 컴퓨팅

최 병 주

1983년 이화여자대학교 수학과 학사. 1988년 Purdue Univ. 전산학 석사. 1990년 Purdue Univ. 전산학 박사. 1995년~현재 이화여자대학교 컴퓨터학과 교수. 관심분야는 소프트웨어공학, 소프트웨어 테스트, 소프트웨어 및 데이타 품질 측정, 소프트웨어 프로세스, 임베디드 시스템 테스팅, 서비스 기반 아키텍처