4

u-City 개발을 위한 설계 패턴

동국대학교 ▌서광익* · 최은만**

1. 서 론

정보통신 분야에서 서비스라는 개념은 전통적 서비스에서 인터넷 기반 서비스를 거쳐 모바일 서비스 기술 다음으로 유비쿼터스 서비스 개념이 변화해 왔다[1]. 이러한 유비쿼터스는 WiBro, RFID, USN 등과 같은 기술을 도시 공간에 융합하여 생활의 편의 증대와 체계적 도시 관리를 가능하게 하는 u-City라는 개념을 만들어 냈다. 따라서 유비쿼터스 컴퓨팅은 u-City를 구현하데 필수적인 기술이며 유비쿼터스 컴퓨팅 기술 수준이 u-City의 수준을 결정하게 된다.

유비쿼터스 컴퓨팅의 궁극적인 목적은 사람과 유사하게 판단하고 행위할 수 있는 컴퓨팅 장치를 만드는 것이다. 유비쿼터스 컴퓨팅을 위해서는 컴포넌트, 센서, 프로세서, 통신, 인터페이스, 보안 등과 관련한 기술이 핵심이다[2]. 유비쿼터스 장치를 개발할 때 반복되는 작업을 패턴으로 정리하여 설계 단계에서 이를 이용해 개발비용과 유지보수 비용을 낮추려는 것이 설계 패턴에 대한 연구이다.

설계 패턴은 목적에 따라 반복적으로 나타나는 일관성을 도면에 일정한 형태나 양식 또는 유형으로 명시하는 것을 말한다. 예를 들어 오랜 기간 동안 도서관을 설계해 온 숙련된 건축 설계자가 있다고 하자. 그 설계자가 반복적인 설계를 하면서 얻은 노하우를 잘 정리하면 다른 설계자도 도서관을 설계 할 경우 참조하여 시간과 비용을 줄일 수 있을 것이다. 이와 같이 소프트웨어 설계 분야에서도 전문가의 경험을 바탕으로 프로그램 개발에 자주 등장하는 문제에 대해 축적해 놓은 지식을 소프트웨어 설계 패턴이라 한다. 이러한 설계 패턴은 설계 시 지침이 되어 설계 초보자도 일정 수준의 설계를 할 수 있도록 한다.

설계 패턴의 개념은 1979년 Alexander가 건축물 설계에 적용하면서 알려지기 시작했다[3]. 소프트웨어 설계는 1995년에 재사용을 중심으로 한 객체지향 소프트웨어의 설계 패턴[4] 이후 많은 관심을 갖게 되었다. 소프트웨어의 규모가 커지고 개발 기술이 빠르게 변하면서 설계 패턴을 적용하는 연구의 분야도 다양화 되어 가고 있다. 특히 재사용과 견고성 중심의 소프트웨어 개발에 필요한 패턴에서 벗어나 인간과 컴퓨터 장치간의 상호작용에 대한 패턴이나 네트워크를 위한 컴퓨터 장치간의 상호작용에 대해서도 패턴 연구가 진행되고 있다.

최근에는 사용자가 다양한 컴퓨팅 서비스들을 그들의 물리적 환경 내에서 언제 어디서나 상황에 맞게 접근할 수 있도록 하는 유비쿼터스에 대한 연구가 활발하다. 이에 따라 사용자와 컴퓨팅 장비와의 상호 작용에 대한 사용자 인터페이스 중심의 설계 패턴 연구가 진행되고 있고, 물리적 환경 내에서 유비쿼터스 장치들 간의 효율적 통신을 위한 연구가 있다. 본 고에서는 이와 같은 최근 연구들에서 제시한 연구 내용과 적용 분야를 정리한다. 2장에서는 최근 설계 패턴에 대한 연구가 다양해지고 있는 가운데 유비쿼터스 컴퓨팅을 고려한 기본적인 설계 패턴 개념에 대해 설명한다. 그리고 3장에서 연구된 구체적인 유비쿼터스 컴퓨팅 지원 패턴을 정리한다.

2. 유비쿼터스 컴퓨팅 설계 패턴

2.1 설계 패턴의 진화

설계 패턴의 개념은 건축물 설계 내용을 재사용함에 따라 얻을 수 있는 이점을 소개한 Alexander의 주장에서 비롯된다[3]. 그는 건축물의 설계에 빈번하게 발생하는 동일 설계내용이 있으며 이런 것들을 하나의 패턴으로 보고 다른 건축물 설계에 재사용하면 많은 이득을 볼 수 있다고 했다. 이와 마찬가지로 소프트웨어를 제작하는데 있어 설계하는 과정에서 기존의 여러 소프트웨어 엔지니어에 의해 검증된 설계를 재사용하여 다양한 효과를 기대하면서 연구되어 왔다. Ward Cunningham과 Kent Beck은 1987년 윈도

* 학생회원
** 종신회원

우 기반 사용자 인터페이스에 대한 다섯 가지의 패턴을 제시했다[5]. 그리고 Erich Gamma와 Richard Helm 등은 1995년에 Gof(Gang of Four)의 책으로 잘 알려져 있는 재사용 가능한 객체지향 소프트웨어에 대한 설계 패턴을 소개하면서 소프트웨어공학 측면의 설계 패턴에 대한 연구가 활발해졌다[4]. 2001년에는 소프트웨어 엔지니어와 HCI(Human Computer Interface) 엔지니어, 도메인 엔지니어들 사이에 발생하는 의사소통 문제를 해결하기 위한 패턴 접근 방법을 제안했다[6]. 2003년 Graham은 웹을 사용하는데 있어 사용성(Usability)을 높이기 위한 패턴을 소개했다[7]. 그 외에도 엔터프라이즈 응용 아키텍쳐의 패턴[8]이나 시험을 위한 패턴[9]등 다양한 분야에서 연구하고 있다.

설계 패턴은 이와 같이 다양한 분야에서 도입하여 활용하고 있고 현재는 유비쿼터스 컴퓨팅을 지원하는 패턴도 활발히 연구 중이다. 하지만 유비쿼터스 컴퓨팅에 패턴을 적용하기 위해서는 유비쿼터스의 특수한 환경을 이해하는 것이 먼저 필요하다.

2.2 유비쿼터스 컴퓨팅을 위한 설계 패턴

유비쿼터스 컴퓨팅은 일반적인 소프트웨어 설계와 커다란 차이점이 있다. 먼저 유비쿼터스 환경에서는 수많은 불특정 장치들이 언제든지 서로 연결될 수 있다는 것이다. 그리고 이러한 장치들은 서로 다른 타입의 하드웨어와 소프트웨어들로 구성되어서 입력과 출력의 형태도 다를 수 있다. 게다가 고정적인 장치와 유동적인 장치, 유선 또는 무선 네트워크 그리고 특정 시점에서 연결이 되어있거나 연결을 시도하려는 특징 등이 있다. 이와 같은 환경을 고려하여 유비쿼터스 컴퓨팅 설계 패턴에 대한 관심 있는 질문을 정리하면 다음과 같다.

- 주변 환경 내에서 접속하고 있으면서 상황정보에 기민하게 대처해야하는 유비쿼터스 컴퓨팅을 위한 새로운 설계 패턴은 무엇인가?
- 이러한 패턴을 주어진 시간과 비용으로 어떻게 검증할 것인가?
- 그 중 실용적인 것과 수정해야 할 것은 어떤 것인가?
- 패턴을 사용하기 위한 절차는 어떻게 정의할 것인가?

유비쿼터스 컴퓨팅 환경을 지원하기 위해서는 장치 스스로가 자신의 위치와 주위 상황을 인지하여 어떤 행위를 해야 하는지 결정할 수 있어야 할 뿐 아니라

그림 1 상황 인지 입출력 패턴 예

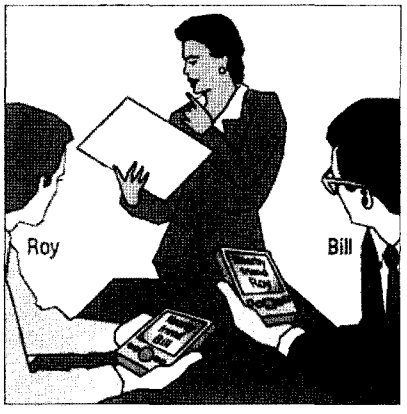

그림 2 물리적 가상 결합 패턴 예

주위 장치와 유동적인 연결이 필수적이다[10]. 이러한 컴퓨팅을 지원하기 위한 상황 인지 입출력(Context-Sensitive I/O) 패턴과 물리적 가상 결합(Physical-Virtual Associations) 패턴을 James와 Gaetano가 개괄적으로 정리하였다[11].

상황 인지 입출력 패턴은 유비쿼터스 컴퓨팅 장치가 어떠한 장소와 상황에서도 사용자의 상태를 파악하고 각 상황에 따라 사용자와의 입출력 인터페이스를 적절히 변환하는 것을 지원한다. 그림 1과 같이 모바일 단말기가 사용자의 상황을 인지하여 운전 중일 때는 벨 소리 모드가 작동하고, 회의 중일 때는 진동 모드로 변환해야 한다. 이를 지원하기 위해서는 단말기가 반드시 사용자의 정황을 인지한 후 입력과 출력의 형식을 사용자 환경에 맞추어야 한다.

물리적 가상 결합 패턴은 유비쿼터스 컴퓨팅 장치 간 연동이 필요한 각 장치들을 인식하고 연결하는데 적은 시간만을 사용하도록 지원한다. 그림 2와 같이 단말기를 이용하여 그룹으로 회의를 하는 경우 가까이 있는 장치들 간의 연결이 쉬워야하고, 새로운 연결을 생성하여 세션이 지속되는 동안 서로의 정보를 공유할 수 있도록 해야 한다.

3. 유비쿼터스 환경에서의 설계 패턴

유비쿼터스의 컴퓨팅을 위한 패턴을 찾기 위해서는 여러 각도에서 사례나 적용 분야를 찾아 개념을

정립하고 정리하는 작업이 필요하다. E. Chung 등은 유비쿼터스 컴퓨팅이 사용되는 목적이나 환경에 따라 패턴의 개념을 총망라해 정리하고 있다[12]. 사무실 환경에서 장치들이 서로를 인식하고 통신할 수 있어야 하는 비즈니스나 응급 상황대처 또는 스마트 홈이나 교육 등 장르별 패턴이 있다. 그리고 유비쿼터스 환경에서 중요시 대두되는 기술인 개인 정보 보호나 장치 식별, 데이터 공유, 위치 인식, 하드웨어 자원 관리 등에 대해 추상 수준별로 정리한 패턴이 있다.

최근 유비쿼터스 컴퓨팅을 지원하기 위한 설계 패턴의 연구 방향은 크게 두 가지라 할 수 있다. 첫째는 사용성 향상을 위한 사용자 인터페이스 관련 패턴이다. 장치는 다양한 사용 목적과 환경에 따라 개발되기 때문에 물리적인 외형 뿐 아니라 사용자가 사용하는 인터페이스 유형도 달라진다. 장치 사용 목적과 환경을 고려하여 패턴을 분류하고 여러 수준의 설계자들을 통하여 설계의 효율성을 측정하거나 효율성을 증대할 수 있는 도구에 대한 연구가 발표되었다. 또 다른 유형은 유비쿼터스 컴퓨팅 특징을 고려한 하드웨어나 네트워크 자원의 설계를 지원하기 위한 연구가 있다. 실시간 최적화를 위한 적응형 시스템이나 센서 네트워크 또는 센서 운영체제 플랫폼을 지원하기 위한 패턴을 제안하고 있다.

3.1 사용자 인터페이스 설계 지원

패턴은 설계에 대한 대상이나 지침 또는 경험적 노하우 등과 같은 지식에 의해 여러 가지 형태로 표현된다. 그 중 상호작용(interaction) 설계를 위한 45개의 패턴을 그룹 A부터 D까지 네 그룹으로 분류하여 설계자들을 통한 경험적 연구가 있다[13]. 그룹 A는 퍼스널 또는 단체별 환경을 지원하거나 사용 장소를 인식하는 것과 같은 유비쿼터스 컴퓨팅 어플리케이션그룹이다. 두 번째 그룹 B는 물리적 객체들과 저장 공간이 가상으로 서로 병합하기 위한 패턴이다. 그룹 C는 엔드 유저의 개인 보호를 위한 정책과 구현 방법에 대한 패턴 그룹이다. 마지막 네 번째 그룹 D는 센서 장치를 통한 유동적인 연결을 위한 패턴 그룹이다. 이러한 그룹별 패턴을 표 1에 정리했다. 그룹 A는 모두 12개의 패턴으로 구성되어 있는데 그중 A1부터 A4까지 어플리케이션을 설계하는 주요한 네 개의 패턴이 있다. 그리고 나머지 8개의 패턴은 주요 네 개(A1~A4) 패턴을 중심으로 특정한 도메인의 어플리케이션을 위한 패턴이다.

표 1 45개 패턴의 그룹별 분류

A – Ubiquitous Computing Genres	B – Physical-Virtual Spaces	C – Developing Successful Privacy	D – Designing Fluid Interactions
Upfront Value Proposition(A1)	Active Map(B1)	Fair Information Practices(C1)	Scale of Interaction(D1)
Personal Ubiquitous Computing(A2)	Topical Information(B2)	Respecting Social Organizations(C2)	Sensemaking of Services and Devices(D2)
Ubiquitous Computing for Groups(A3)	Successful Experience Capture(B3)	Building Trust and Credibility(C3)	Streamlining Repetitive Tasks(D3)
Ubiquitous Computing for Places(A4)	User-Created Content(B4)	Reasonable Level of Control(C4)	Keeping Users in Control(D4)
Guides for Exploration and Navigation(A5)	Find a Place(B5)	Appropriate Privacy Feedback(C5)	Serendipity in Exploration(D5)
Enhanced Emergency Response(A6)	Find a Friend(B6)	Privacy-Sensitive Architectures(C6)	Context-Sensitive I/O(D6)
Personal Memory Aids(A7)	Notifier(B7)	Partial Identification(C7)	Active Teaching(D7)
Smart Homes(A8)		Physical Privacy Zones(C8)	Resolving Ambiguity(D8)
Enhanced Educational Experiences(A9)		Blurred Personal Data(C9)	Ambient Displays(D9)
Augmented Reality Games(A10)		Limited Access to Personal Data(C10)	Follow-me Displays(D10)
Streamlining Business Operations(A11)		Invisible Mode(C11)	Pick and Drop(D11)
Enabling Mobile Commerce (A12)		Limited Data Retention(C12)	
		Notification on Access of Personal Data(C13)	
		Privacy Mirrors(C14)	
		Keeping Personal Data on Personal Devices(C15)	

표 1에 정리된 패턴은 유비쿼터스 컴퓨팅의 응용 분야를 폭넓게 분류한 것이다. T. S. Saponas 등은 이를 참고하여 연구 범위를 미래의 디지털 홈으로 제한하여 패턴을 정의했다[14]. 디지털 홈과 유비쿼터스 컴퓨팅 그리고 그와 관련된 학술자료와 연구 및 상품들을 수집한 후 공통된 주제와 디자인들을 식별하여 48개의 디자인 패턴을 정리했다. 각 패턴은 모두 사용자 수행능력 보조(Aiding Human Abilities), 그룹 커뮤니케이션(Group Communication), 무제한 지역 접근성(Location-Free Access), 상황별 행동 인식(Activity Awareness), 개인 정보 보호(Privacy), 사용성(Usability)과 같이 6개의 그룹으로 분류된다. 그림 3은 A5에 해당하는 재고(Inventory) 패턴의 배경 이미지이다. 재고 패턴은 RFID를 통해 주방에 있는 음식 재료의 종류와 위치들을 알 수 있도록 지원하는 사용자 수행능력 보조 그룹에 속한다. 표 2에서는 재고 패턴의 배경과 개괄적인 내용을 정리했다. 여기서 정의한 모든 패턴은 고유한 식별 번호와 이름, 그리고 문제의 배경을 암시하고 있는 이미지, 개요와 배경, 문제점, 해결방법, 구체적인 사용법과 참고 자료를 제시한다.

설계 패턴은 문제를 해결하기 위한 방법과 모범 사례를 제공하기 때문에 개발 초기에 대략적인 설계와 프로토타입을 생성하는데 도움을 준다. Dasmask는 개발 초기에 미리 정의된 설계 패턴을 이용하여 서로 다른 사용자 인터페이스를 가진 장치들에 대한 상호작용을 프로토타입할 수 있도록 지원하는 도구이다[15].

그림 3 재고 패턴 이미지

표 2 재고 패턴

개요	사용자가 주방에 있는 다양한 물품들의 위치를 쉽게 찾을 수 있도록 재고 시스템을 구현한다.
배경	대부분의 가정에는 다양한 음식 재료를 저장해 놓고 요리를 해서 식사를 한다. 그러나 음식 재료가 너무 많거나 다양해 수납 위치나 유통기한을 넘긴 재료를 찾기 어렵다.
문제	모든 음식 재료의 위치와 유통기한을 기억하기 어렵다.

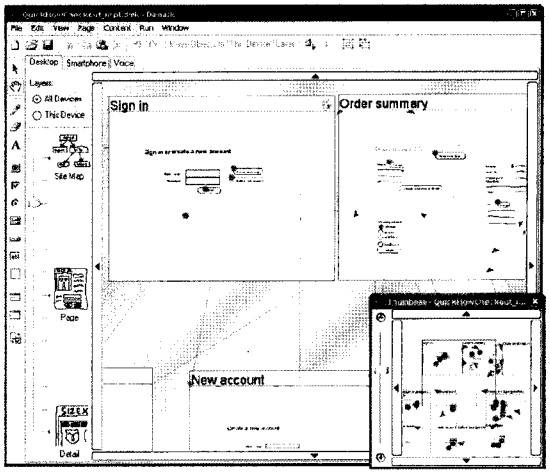

그림 4 Dasmask 사용자 인터페이스

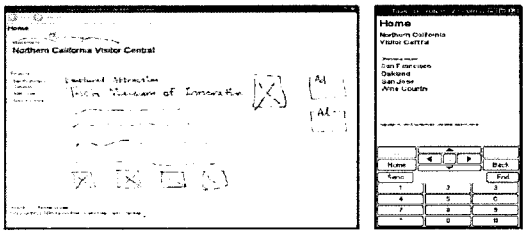

그림 5 PC(좌)와 이동전화(우)의 화면

PC에서의 웹 스타일 인터렉션이나 스마트 폰에서의 웹 스타일 인터렉션 그리고 이동전화나 PC에서의 음성 UI 등이 Dasmask에서 제공하고 있는 프로토타입의 세 가지 유형이다. 그림 4는 Dasmask의 사용자 인터페이스이다. 설계자는 Dasmask에 있는 패턴과 레이어를 이용하여 설계하고자 원하는 장치를 위한 UI를 설계한다. 그리고 설계한 장치와 연결하여 통신해야 하는 또 다른 장치를 같은 방법으로 설계한다. 그림 4는 사용자가 사용할 수 있는 UI들 간의 관계를 Dasmask가 지원하는 패턴으로 설계한 것이다. 그리고 그 결과 그림 5와 같이 PC와 스마트 폰 간의 통신 이후 각 단말기에서 사용자에게 출력해주는 UI를 동기화해서 보여준다.

3.2 네트워크 및 시스템 설계 지원

설계 패턴을 사용하여 효율성을 증대시키려는 노력은 점점 증가하고 있고 유비쿼터스 컴퓨팅도 예외가 아니다. 유비쿼터스 환경을 구축하기 위해서는 사용되는 목적과 장치가 다양하면서 특수한 만큼 적용되는 기술 또한 복잡하고 복합적이다. 특히 유비쿼터스 컴퓨팅을 지원하기 위해서는 산재된 노드들 간의 통신을 위한 센서 네트워크 어플리케이션과 이를 효과적으로 지원하는 TinyOS 기술이 필수적이다. 특히 센서와 센서 네트워크의 사용 증가는 개발 실패를

줄이기 위해 경험자들의 지식과 기술을 패턴으로 정리하는 것이 필요하다. 센서 네트워크를 위해 연구된 패턴은 관찰자(Observer) 패턴과 프록시(Proxy) 패턴 그리고 중재자(Mediator) 패턴이 있다[16].

기상 예보 시스템을 구축하기 위해 온도, 습도, 기압 등을 측정하는 수 천 개의 다목적(multipurpose) 센서 또는 다중(multiple) 센서가 쓰이는 시스템을 구축한다고 하자. 각 센서가 하나의 OS와 직접 통신하도록 설계한다면 일대다 관계에서 많은 부하가 생길 수 있다. 이 때 관찰자 패턴을 이용한다.

그림 6은 각 센서가 직접 OS의 행동을 검사하고 있는 설계에서 서브젝트(Subject)가 OS의 상태를 검사하고 각 노드들에게 알려주는 역할을 함으로 OS의 부하를 줄여준다.

그리고 인터넷을 통해 센서 노드를 접근해야 하는 환경에서는 그림 7과 같이 프록시를 통해 센서를 구성하고 있는 각 모달(modal)을 접근하도록 하여 통신 부하를 최소화 한다. 프록시 패턴은 센서 노드나 센서 노드에 있는 객체 또는 기타 자원들의 접근을 제어할 때 사용할 수 있다. 그리고 객체의 인스턴스를 생성하는데 오랜 시간이 소요되는 등 많은 자원을 요구하는 객체의 접근을 프록시가 제어하여 네트워킹의 효율성을 높일 수 있다.

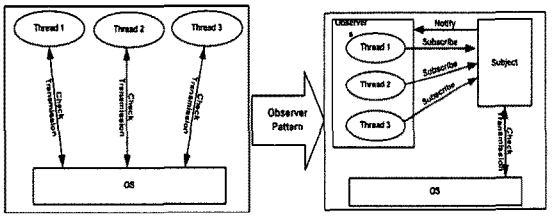

그림 6 관찰자 패턴

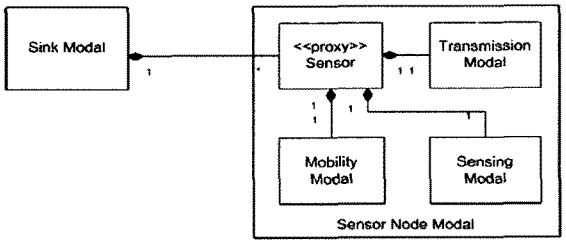

그림 7 프록시 패턴

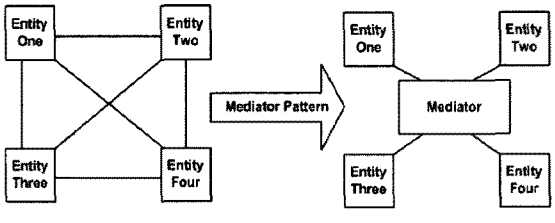

그림 8 중재자 패턴

중재자 패턴은 그림 8과 같이 복잡한 통신을 하는 개체들 사이에 복잡한 통신을 관할하는 중재자를 두는 것이다. 이렇게 하면 시스템 관리자는 중재자와 관련 모듈을 독립적으로 관리할 수 있어서 유지보수성도 높이고 새로운 노드나 기능을 추가할 경우에도 중재자에만 접속 하도록 하기 때문에 다른 노드에 영향을 미치지 않고 쉽게 추가할 수 있다. 그리고 다대다에서 일다대(one-to-many)의 구조로 통신 프로토콜도 간소화한다. 여기서 중재자는 노드(또는 컴포넌트나 모듈)들 사이에서 통신을 제공하기 위한 인터페이스 역할을 한다. 센서 네트워킹 환경에서 복잡한 통신을 단순하게 할 수 있다. 또한 모듈간의 관계가 복잡하게 얽혀 있어 재사용이 어려울 경우 중재자는 모듈의 결합도를 낮춰 재사용을 높인다.

센서 네트워크 환경을 지원하기 위해서는 제한된 자원과 에너지를 문제 영역에 잘 맞게 운영할 수 있는 TinyOS가 필요하다. 이러한 TinyOS는 여러 목적으로 사용할 수 있는 컴포넌트의 라이브러리이기 때문에 각 컴포넌트의 관계와 기능을 분석하여 패턴을 찾을 수 있다.

퍼싸드(Facade) 패턴이나 디스패쳐(Dispatcher) 패턴 등은 코드 재사용성과 개발 복잡도를 줄이기 위한 패턴이다[17,18]. 파일 시스템이나 네트워킹과 같은 시스템 컴포넌트는 수많은 컴포넌트가 서로 복잡한 관련성을 갖는다. 이러한 경우 그림 9와 같이 퍼싸드 패턴을 사용하여 인터페이스를 제공해서 사용자가 원하는 기능을 쉽게 이용할 수 있도록 한다. 독립적인 여러 컴포넌트들이 모두 다른 추상화 수준으로 구현되거나 특정 부분보다는 시스템의 전체적인 관점에서 추상화할 경우에 사용할 수 있다.

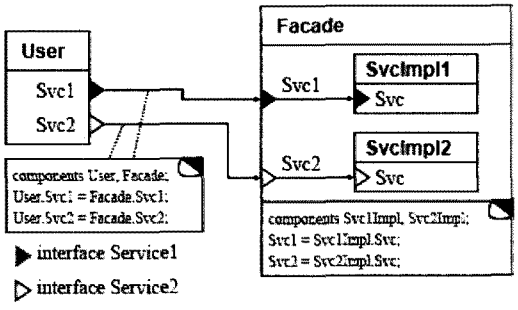

그림 9 퍼싸드 패턴

```
configuration Matchbox {
provides {
interface FileDelete;
interface FileDir;
interface FileRead[uint8_t fd];
```

```
interface FileRename;
interface FileWrite[uint8_t fd];
}
}
implementation {
components Read, Write, Dir, Rename, Delete;
FileDelete = Delete.FileDelete;
FileDir = Dir.FileDir;
FileRead = Read.FileRead;
FileRename = Rename.FileRename;
FileWrite = Write.FileWrite;
}
```

위 소스코드는 TinyOS를 구현하는데 사용하는 C 기반 언어의 인터페이스와 동시성(concurrency) 모델을 제공하는 necC[19]을 이용하여 퍼싸드 패턴을 구현한 예이다. 파일을 쓰고 읽고 지우고 새 이름으로 저장하는 일을 하는 Matchbox 파일링 시스템이 인터페이스를 단일하게 추상화하여 표현하기 위해 퍼싸드 패턴을 사용하고 있다. 파일을 다루는 오퍼레이션은 모든 컴포넌트에서 구현되지만 인터페이스를 이용해 단 하나만 정의했다. 그리고 각각의 컴포넌트는 각자의 추상 수준에 맞도록 구현하고 있는 것을 볼 수 있다. 그림 9에서 동일한 Svc를 사용자 Svc1와 Svc2에 맞도록 SvcImp1, SvcImp2로 각각 구현하여 제공하고 있는 것과 같은 의미이다.

네트워크 상의 한 노드가 여러 종류의 활성 메시지(Active Messages)를 받고 각 메시지에 따라 특정 컴포넌트를 호출해야 한다면 그림 10과 같이 디스패쳐(Dispatcher) 패턴을 사용할 수 있다. 이렇게 함으로써 Op1이나 Op2로 표현된 각 기능을 독립적으로 구현하여 추가하거나 수정하여 재사용이 가능하도록 지원한다.

다음은 디스패쳐 패턴의 예이다. AddXY가 OperatinX와 OperationY를 추가하기 위해 Dispatcher의 인터페이스를 이용하고 있는 nesC 예제 코드이다.

동시 접속한 다수의 센서가 각자 특정 서비스에 대한 인스턴스를 모두 가지고 있어야하는 경에 각인스턴스를 효율적으로 관리할 수 있는 서비스 인스턴스(Service Instance) 패턴이 있다. TinyOS는 연결되어 있는 센서의 접속 상태를 관리하기위해 접속 시간을 체크하는 타이머가 센서의 개수만큼 필요할 경우 서비스 인스턴스 패턴을 사용할 수 있다. 그림 11은 서비스 인스턴트 패턴의 예이다.

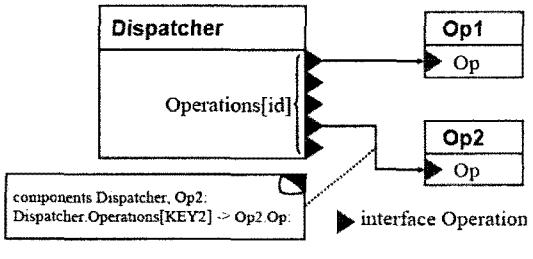

그림 10 디스패쳐 패턴

```
configuration Dispatcher {
uses interface Operation as Op[uint8_t id];
}
configuration AddXY {}
implementation {
components Dispatcher, OperationX, OperationY;
Dispatcher.Op[OP_X] -> OperationX.Op;
Dispatcher.Op[OP_Y] -> OperationY.Op;
}
```

각 인스턴스가 독립적이면서 컴포넌트가 각 서비스에 대한 다중 인스턴스(multiple instance)를 제공해야 하는 경우 서비스 인스턴스 패턴을 사용할 수 있다. 그림 11은 ServiceProvider가 ResourceC를 필요로 하는 사용자에게 각각 독립적으로 서비스를 제공하고 있다.

```
module TimerM {
provides interface Timer[uint8_t id];
uses interface Clock;
}
implementation {
enum {
NUM_TIMERS = uniqueCount("Timer")
}
timer_t timers[NUM_TIMERS];
command result_t Timer.start[uint8_t id](...){}
}
```

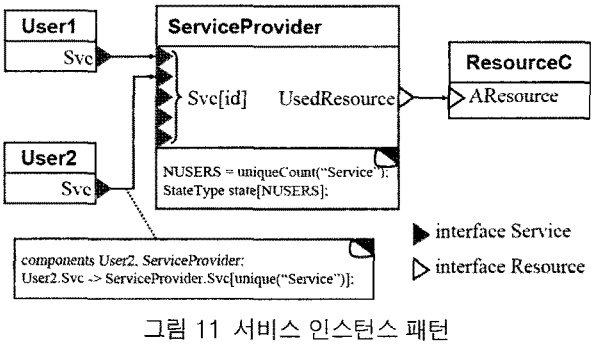

그림 11 서비스 인스턴스 패턴

예제 코드를 보면 TimerM은 몇 개의 타이머를 할당할 것인지 판단하기 위해 고유 ID를 사용하는 uniqueCount를 가지고 있다. 만약 TimerC가 자기의 고유 타이머가 필요하다면 서비스 제공자(Service-Provider)인 TimerM을 "C.Timer -> TimerC.Timer [unique("Timer")];"와 같이 이용하는 코드를 작성하면 된다.

유비쿼터스 컴퓨팅에서 애드혹(Ad-hoc) 네트워크상의 노드들은 무선 인터페이스를 사용하여 서로 통신하고, 멀티 홉 라우팅 기능에 의해 무선 인터페이스가 가지는 통신 거리상의 제약을 극복하며, 노드들의 이동이 자유롭기 때문에 네트워크 토폴로지가 동적으로 변화되는 특징이 있다. 따라서 TinyOS는 여러 라우팅 알고리즘을 가지고 있는데, 상황에 따라 독립적으로 알고리즘을 선택할 수 있도록 지원해 주는 패턴이 Placeholder 패턴이다.

Placeholder는 그림 12와 같이 컴포넌트 User1이나 User2가 원하는 서비스를 간접적으로 표현한다. 그리고 각 컴포넌트는 서비스가 필요할 때 Placeholder를 통해 원하는 서비스를 사용한다. 네트워크 레이어에서 어플리케이션 수준에서는 센서들을 확인하고 통신이 가능하도록 구성하는 애드혹(ad-hoc) 라우팅이 필요하다. 그러나 어플리케이션 설계는 특정 라우팅을 구현하는 추상 수준과 독립적으로 수행된다. 이러한 레이어 설계는 라우팅 알고리즘을 개선하거나 새로 추가하는 작업을 쉽게 해 준다. 그림 12에서는 Placeholder가 각 레이어를 구분해 주는 경계가 되는 것이다. 즉 Placeholder가 인터페이스를 제공하여 라우팅을 구현하고 있는 서브시스템을 대신하는 것이다.

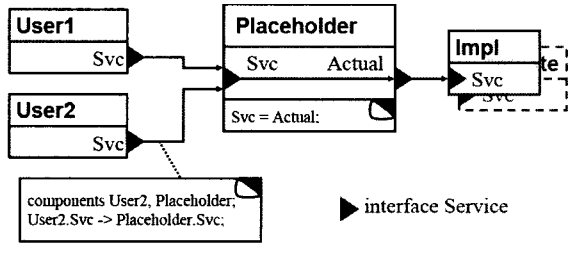

그림 12 Placeholder 패턴

```
configuration CollectionRouter
provides {
interface StdControl
interface SendMsg
}
uses {
interface StdControl
interface SendMsg
}
}
implementation {
SC = ActualSC;
Send = ActualSend;
}
```

예제 소스에서 CollectionRouter는 인터페이스를 제공하고 있다. 그리고 라우팅을 사용하려는 컴포넌트는 다음과 같이 CollectionRouter를 연결하고 있어야 한다.

```
configuration Sensing { ... }
implementation {
components SensingM, CollectionRouter;
SensingM.Send = CollectionRouter.Send;
...
}
```

그리고 어플리케이션이 라우팅 컴포넌트를 선택하기 위해서는 다음과 소스코드와 같이 CollectionRouter를 사용한다.

```
configuration AppMain { }
implementation {
components CollectionRouter, EWMARouter;
CollectionRouter.ActualSC -> EWMARouter.SC;
CollectionRouter.ActualSend -> EWMARouter.Send;
...
}
```

4. 결 론

다양한 엔지니어링 분야에서 설계 패턴을 기초 원리로 정립하고 사용하려는 노력이 있어 왔다. 그러한 흐름은 최근 각광받는 연구의 한 분야인 u-City의 기반 기술인 유비쿼터스 컴퓨팅에도 예외가 아니다. 유비쿼터스의 단말기나 노드들은 제한된 자원과 사용 목적에 따라 사용자 인터페이스나 구현 기술이 모두 특성화될 수밖에 없다. 따라서 u-City 기반의 유비쿼터스 환경을 지원하는 설계 패턴도 사용 환경과 구현 기술을 염두하고 정의해야 한다. 이러한 유비쿼터스 특징 중 사용자와 단말장치의 사용 목적과 환경에 따라 하드웨어와 소프트웨어의 설계를 결정짓는 인터페이스 관련 패턴이 있다. 그리고 복잡한 모듈로 구현된 센서 노드와 TinyOS는 모듈의 재사용성과 유

지보수성을 높이기 위한 패턴들이 있다. 이러한 패턴들이 모두 특화된 목적에 따라 정의된 것과 같이 u-City를 구축하는데 필요한 유비쿼터스 컴퓨팅의 적용 목적과 구현 기술의 범위가 넓어짐에 따라 다양한 설계 패턴 연구가 필요할 것이다.

참고문헌

[1] WoongHee Park, WooSoo Jeong, HyangSuk Cho, A Study of the Evolution of the u-City Service, Preceedings of the conference on PICMET2007, pp. 1141-1146, 2007.

[2] Kim, J. Y. Ubiquitous Computing: Business Models and Industry Forecast, Samsumg Economic Research Institute, 2004.

[3] Christopher Alexander, A Pattern Language: Towns, Building, Construction, Oxford Univ. Press, 1977.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and Joh Vlissiders, Design Pattern: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[5] Ward Cunningham and Kent Beck, Using Pattern Languages for Object-Oriented Programs, Technical Report CR-87-43, Tektronix Inc., 1987.

[6] Jan Borchers, A Pattern Approach to Interaction Design, Proceedings of the 3rd conference on Designing Interactive systems, pp 369-378, 2000.

[7] Ian Graham, A Pattern Language for Web Usability, Addison Wesley, 2003.

[8] Martin Fowler, Patterns of Enterprise Application Architecture, Addison Wesley, 2003.

[9] Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code, Addison Wesley, 2007.

[10] Ubiquitous Compuing Grand Challenge, http://www-dse.doc.ic.ac.uk/projects/UbiNet/GC/index.html.

[11] James Landay, Gaetano Boriello, Design patterns for Ubiquitous Computing, Computer, Volume 36, Number 8, pp 93-95, 2003.

[12] Chart of Patterns in Ubiquitous and Context Computing UbiComp Application Genres, http://kettle.cs.berkeley.edu/ubicomp/2.

[13] E. S. Chung, J. I. Hong, J. Lin, M. K. Prabaker, J. A. Landay, A. L. Liu, Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing, Proceedings of the 5th conference on Designing Interactive systems, pp. 233-242, 2004.

[14] T. S. Saponas, M. K. Prabaker, G. D. Abowd, J. A. Landay, The Impact of Pre-Pattens on the Design of Digital Home Applications, Proceedings of the 6th conference on Designing Interactive systems, pp. 189-198, 2006.

[15] James Lin, James A. Landay, Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces, Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, pp. 1313-1322, 2008.

[16] S. O. Amin, M. S. Siddiqui, and Choong Seon Hong, Building Scalable and Robust Architecture for Ubiquitous Sensor Networks with the help of Design Patterns, Proceeding of the Conference on Multimedia and Ubiquitous Engineering, pp. 1046-1050, 2007.

[17] D. Gay, P. Levis, and D. Culler, Software Design Patterns for TinyOS, Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, pp. 40-49, 2005.

[18] P. Levis and D. Gay, TinyOS design Patterns, http://www.cs.berkeley.edu/~pal/tinyos-patterns, 2004.

[19] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler, The nesC language: A holistic approach to networked embedded systems, conference on PLDI'03, 2003.

서 광 익

2002 동국대학교 컴퓨터공학 학사
2004 동국대학교 컴퓨터공학 석사
2006 동국대학교 컴퓨터공학과 박사 수료
관심분야 : 소프트웨어 테스트, 소프트웨어 품질, 프로세스
E-mail : bradseo@dongguk.edu

최 은 만

1982 동국대학교 전산학과 학사
1985 한국과학기술원 전산학과 석사
1993 일리노이 공대 전산학과 공학박사
1985~1988 한국표준연구소연구원
1988~1989 데이콤 주임연구원
1998~2004 한국정보과학회 소프트웨어공학연구회 운영위원
2000, 2007 콜로라도 주립대 전산학과 방문교수
2002 카네기멜론대학 소프트웨어공학 과정 연수
1993~현재 동국대학교 컴퓨터공학과 교수
관심분야 : 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension
E-mail : emchoi@dgu.ac.kr