상황정보 기반 자기적응형 소프트웨어 설계 방법*

황성진O, 박준석, 문미경, 염근혁
부산대학교 컴퓨터 공학과
{hssj4543O ,pjs50, mkmoon, yeom}@pusan.ac.kr

An Approach for Designing Self-Adaptive Software based on Context Information

Seongjin HwangO, Joonseok Park, Mikyeong Moon, Keunhyuk Yeom
Department of Computer Engineering, Pusan National University

요 약

최근 유비쿼터스 컴퓨팅 환경의 실현 가능성이 높아지면서 동적으로 변화하는 외부 환경에서의 소프트웨어 역할이 중요해지고 있다. 유비쿼터스 환경의 소프트웨어는 다양한 센서로부터 입력되는 문맥정보를 분석하고 그 결과에 따라 적절하게 서비스를 제공할 수 있는 자기적응형(self-adaptive) 소프트웨어 형태가 되어야 한다. 이러한 특징을 가진 소프트웨어를 개발하기 위해서는 문맥정보에 대한 정적분석 활동과 문맥 변화에 상호 작용하는 동적분석 활동이 개발 전 과정에 걸쳐 체계적으로 수행되어야 한다. 본 연구에서는 외부 환경의 문맥정보에 가변적으로 반응하는 자기적응형 소프트웨어의 요구사항을 분석하고, 문맥정보 조건에 따라 재구성 가능한 컴포넌트 기반 아키텍처를 설계하기 위한 자기적응형 소프트웨어 설계 방법을 제시한다. 또한 본 연구의 방법을 적용하여 설계한 스마트 홈 시스템에 대한 사례연구를 소개한다.

1. 서 론

유비쿼터스 컴퓨팅은 사용자의 상황에 따라 제공되는 서비스가 변해야 한다. 유비쿼터스 환경을 구축하기 위해서는 지능화된 초소형 센서가 필요하고, 이를 통하여 상황을 인지하여 사물과 주변 환경의 변화를 인식 또는 추적하여 그에 따른 적절한 정보와 서비스를 제공하여야 한다[1]. 유비쿼터스 컴퓨팅 환경의 실현 가능성이 높아지면서 동적으로 변화하는 외부 환경에서의 소프트웨어 역할이 중요해지고 있다. 이러한 소프트웨어는 외부 환경의 변화에 직면하였을 때 동작을 멈추는 것이 아니라, 변화를 감지하고 대안을 선택하여 지속적으로 서비스를 제공할 수 있는 자기적응형 (self-adaptive) 소프트웨어가 될 필요가 있다. 자기적응형 소프트웨어란 자신의 행위를 평가하여 소프트웨어가 원래 의도한 것을 수행하지 못하거나, 더 나은 기능을 수행하는 것이 가능하다면 스스로 행위를 변경할 수 있는 소프트웨어를 말한다[2]. 자기적응형 소프트웨어는 변화하는 외부 환경에 대응하여 자신의 행동을 변경하기 위해서 변화하는 외부 환경에 대한 정보를 이용하게 된다. 소프트웨어가 동작하는 외부 환경을 문맥(context)이라 하는데 자기적응형 소프트웨어는 문맥의 변화에 따라 기능이 달라진다. 따라서 자기적응형 소프트웨어를 개발하기 위해서는 소프트웨어의 기능을 분석하고 설계하는 것뿐만 아니라 소프트웨어가 동작하는 외부 환경 즉, 문맥의 변화를 분석하고 설계하는 것이 병행되어야 한다. 그러나 현재 적응형 소프트웨어에 대한 연구는 소프트웨어가 외부 환경의 변화에 ' 어떻게' 적응하도록 할 것인가에 대한 적응 메커니즘에 초점이 맞추어져 있으며, 적응형 소프트웨어에 영향을 주는 문맥정보를 모델링하는 기법이나 적응을 위해 대체할 수 있는 기능들을 찾아내는 방법에 대한 연구가 부족한 실정이다. 또한 기존의 소프트웨어 분석 설계 방법들은 소프트웨어를 개발하기 위하여 각 단계에서 수행하는 세부 활동과 산출물을 제시하고 있으나 소프트웨어가 동작하고 반응해야 하는 문맥에 대한 분석은 초기 단계에서만 간단히 수행되고 그 이후의 단계에서는 문맥의 변화가 적절히 반영되지 못하고 있다. 기존의 소프트웨어와 달리 자기적응형 소프트웨어는 문맥과 밀접하게 상호 작용하기 때문에 소프트웨어 분석 설계 방법의 전 과정에 문맥에 대한 분석, 설계 정보가 반영되어 있어야 한다.

따라서 본 논문에서는 소프트웨어 설계 방법의 전 과정에 걸쳐 문맥의 변화에 따른 자기적응형 소프트웨어의 행동의 변화를 분석하고 설계할 수 있는 자기적응형 소프트웨어 설계 방법을 제시한다. 이 방법은 요구사항 분석 단계에서부터 자기적응형 소프트웨어의 동작에 영향을 주는 문맥과 그 문맥에 반응하는 기능을 추출한다. 또한 이를 분석하여 동적으로 적응이 가능한 컴포넌트 기반 아키텍처를 개발하며, 이 아키텍처를 구성하는 컴포넌트와 커넥터를 설계한다. 본 연구 방법은 스마트 홈 시스템 개발에 적용되었으며, 그 결과를 본 논문에 방법론과 같이 기술한다.

2. 관련 연구

2.1 문맥 관련 연구

* 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10536-0)지원으로 수행되었음.

문맥은 다양할 뿐 아니라 정해진 표현 방법이 없기 때문에 자기적응형 소프트웨어가 문맥 정보를 이용하여 변화하는 외부 환경에 적응하도록 하는 것은 어려운 일이다[3]. Dey 와 Abowd 의 문맥에 대한 정의에 따르면, " 문맥이란 사용자와 어플리케이션 사이의 상호 작용에 고려되어야 할 사람, 장소, 사물의 상황을 설명하는 정보" 를 나타낸다[4]. Dey 와 Abowd 는 이러한 문맥 정의를 바탕으로 문맥 인식 컴퓨팅 분야의 연구를 진행하여 Context Toolkit 을 개발하였다. Context Toolkit 은 문맥 인식 가능한 어플리케이션의 개발을 지원하기 위해서 센서로부터 문맥 정보를 받아서 이를 처리하는 Widget, Interpreter, Aggregator 라는 추상화된 컴포넌트를 제공한다. 개발자는 Context Toolkit 을 이용하여 문맥 처리에 필요한 일반적인 메카니즘을 제공받아 좀 더 쉽고 효율적으로 문맥 인식 애플리케이션을 개발할 수 있다[5]. 그러나 문맥을 추출하고 모델링하여 구체적으로 표현하는 방법을 제공해주지는 못하고 있다[6].

2.2 자기적응형 소프트웨어

자기적응형 소프트웨어란 자신의 행위를 평가하여 소프트웨어가 원래 의도한 것을 수행하지 못하거나, 더 나은 기능을 수행하는 것이 가능하다면 스스로 행위를 변경할 수 있는 소프트웨어를 말한다[2]. 자기적응형 소프트웨어에 대한 연구로써 MIT(Massachusetts Institute of Technology)의 DDA(Dynamic Domain Architecture)[7], UCI(University of California, Irvine)의 Architecture-based Self-adaptive Approach[8], CMU(Carnegie Mellon University)의 Rainbow Project[9]가 있다.

MIT 의 DDA 는 도메인 내의 모든 애플리케이션이 가지고 있는 공통적인 기능과 공통적인 기능 속에 존재하는 가변성을 표현하는 도메인 아키텍처 개념을 자기적응형 소프트웨어에 적용하고 있다. 그러나 자기적응형 소프트웨어의 적응에 있어서 코드와 알고리즘 대체를 통한 접근 방식을 취하고 있다. UCI 의 Architecture-based Self-adaptive Approach 는 C2 아키텍처 스타일을 바탕으로 외부 환경의 변화를 감지(Detection)하고 변화에 대처할 수 있는 계획(Plan)을 세우고 변경이 필요한가를 평가(Evaluation)하여 변경(Adaptation)을 처리하는 메커니즘을 제공한다. Rainbow Project 는 소프트웨어 시스템의 적응을 위하여 소프트웨어 아키텍처와 재사용 가능한 기반을 사용하는 Rainbow framework 을 제시한다. 즉, 추상화된 아키텍처 모델을 사용하여 현재 동작 중인 시스템의 런타임 속성을 모니터링하고 모델을 평가하여 문제가 발생했을 때 동작 중인 시스템에 적응을 수행한다. 이러한 자기적응형 소프트웨어에 대한 연구들은 소프트웨어가 외부 환경의 변화에 ' 어떻게' 적응하도록 할 것인가에 대한 적응 메커니즘에 초점을 두고 있다. 즉, 적응형 소프트웨어에 영향을 주는 문맥정보를 모델링하는 기법이나 적응을 위해 대체할 수 있는 기능들을 찾아내는 방법에 대한 연구는 부족하다.

3. 자기적응형 소프트웨어 설계 방법

본 논문에서 제시하는 자기적응형 소프트웨어 설계 방법은 그림 1 에서 도식화한 것과 같이 크게 세 단계 -범위 결정 단계, 문맥 기반 요구사항 분석 단계, 문맥 기반 아키텍처 설계 단계- 로 구성된다. 제시되는 개발방법은 문맥에 따른 기능의 변화를 위해 컴포넌트 기반 개발 방법을 따른다. 즉, 문맥의 변화에 따라 컴포넌트가 추가, 삭제, 대체 되기 위하여 컴포넌트 기반의 아키텍처가 설계된다. 본 논문에서 적용하고 있는 스마트 홈 시스템은 거주자의 기본 정보, 행동, 온도와 빛의 세기 등의 환경 문맥정보를 거주 환경에 설치된 센서를 통해 수집, 분석하여, 문맥정보에 맞게 거주자에게 알맞은 서비스를 제공해 주는 홈 오토메이션 구현을 목표로 하고 있다.

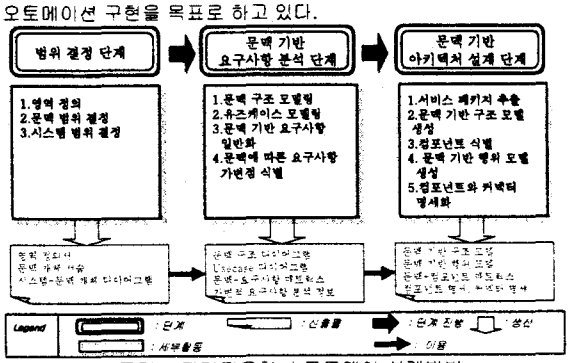


그림 1 자기적응형 소프트웨어 설계방법

3.1 범위 결정 단계

범위 결정 단계에서는 소프트웨어가 동작하는 문맥을 정의하고, 소프트웨어의 시스템 범위와 문맥의 범위를 결정한다.

■ 영역 정의

영역 정의에서는 개발하고자 하는 소프트웨어와 소프트웨어가 동작하는 환경인 문맥을 자연어로 기술하여 영역 정의서를 만든다. 또한 문맥의 기술적인 제약사항, 서비스 제약사항 등을 같이 기술한다. 표 1은 스마트 홈 시스템 개발을 위한 영역 정의서이다.

표 1 스마트 홈 시스템의 영역 정의서

시스템 정의	
u-All home	가옥 구성요소들을 통폭하고, 이에 따라 개별적인 서비스가 사용할 수 있도록 제어하는 환경을 구축하는 유비쿼터스 환경에서의 Smart Home이다.
문맥 정의	
Intelligent Home	집안 요소요소에 설치된 센서들에 의해서 가옥 구성원 개개인에게 개별적인 서비스가 제공되는 유비쿼터스 환경의 가장 중심으로 Room1, Room2, Room3, Living Room, Kitchen, Bathroom, Verandal, Verandal로 구성되어 있다.
기술적 제약사항	
1. 센서들은 무선으로 연결되어 있다.	
서비스 제약사항	
1. 설정된 문맥이 default값은 Main User만이 구성할 수 있다. 2. Private Room에 있는 전자기기에 대해서는 Main User의 경우 주변 전자기기에 대해서는 별도로 설정이 가능하다.	

■ 문맥 범위 결정

소프트웨어 외부에 존재하는 개체들을 추출함으로써 문맥의 범위를 결정한다. 이러한 개체들을 문맥 개체(context Entity)라고 한다. 문맥 개체는 소프트웨어 외부에서 소프트웨어와 상호 작용하며 소프트웨어에 영향을 주는 대상으로 장소, 사람, 사물(물리적 장치, 컴퓨팅 장치)로 분류된다. 각 분류 별 추출된 문맥 개체들은 문맥개체 서술에 기술된다. 표 2 는 스마트 홈의 room1 내에서 추출된 문맥개체들을 기술한 문맥개체 서술서이다.

■ 시스템 범위 결정

소프트웨어는 외부 환경에 존재하는 문맥 개체와 상호작용을 한다. 문맥 범위 결정에서 추출된 문맥 개체와 소프트웨어의

관계를 시스템-문맥 개체 다이어그램으로 모델링함으로써 소프트웨어의 시스템 범위를 결정한다. 그림 2 는 스마트 홈의 시스템-문맥 개체 다이어그램이다.

표 2 스마트 홈 Room1 의 문맥 개체 서술서

Category		Name	Description
Place		Room1	항에 설치된 컴퓨터와 Main User의 개인 컴퓨터가
		Private	Main User의 개인 공간
Person		Main User	개인방으로 사용하는 사용자
		User	컴퓨터에 대해서 일시적인 사용만 가능
		Temporary User	컴퓨터에 대해서 일시적인 사용만 가능
Thing	Physical Thing	Door	출입을 감지하는 센서가 부착
		Window	개폐여부를 감지하는 센서가 부착
		Wall	온도와 습도를 감지하는 센서가 부착
		Roof	외부의 온도에 따라 냉난방을 하는 센서가 부착
	Computing Device	Lighting	조도에 따라서 고명을 켜는 센서
		inDoor/outDoor	문이 열리면 무엇을 기준으로 출입을 감지하는 센서
		inWindow/outWindow	창문이 열리면 무엇을 기준으로 개폐여부를 감지하는 센서
		Temperature / Hygrometer	온도와 습도를 감지하는 센서
		Air conditioner	온도에 따라서 냉난방을 하는 센서

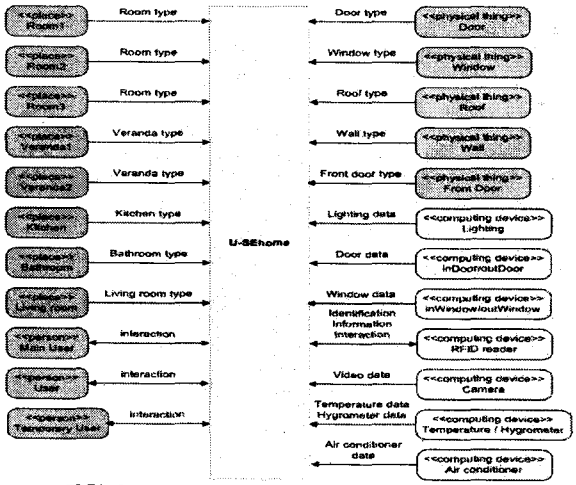


그림 2 스마트 홈 시스템-문맥 개체 다이어그램

3.2 문맥 기반 요구사항 분석 단계

이 단계에서는 문맥 개체간의 관계를 모델링하고, 소프트웨어의 기능을 유즈케이스로 표현한다. 추출된 각각의 유즈케이스에 대해서 소프트웨어와 문맥 개체의 상호작용을 상세화된 수준에서 분석하여 문맥과 문맥 기반 요구사항을 추출하고 이를 일반화한다. 요구사항은 반복적인 분석을 통해 문맥에 따른 가변점이 발견되고 상세화한다.

■ **문맥 구조 모델링**

UML 의 Class 다이어그램을 이용하여 범위 결정 단계(3.1 절)에서 추출된 문맥 개체들을 구조화한다. 문맥 개체의 카테고리는 클래스에 stereotype 으로 표현한다. 이렇게 구조화된 문맥 개체들은 Use case 다이어그램에서 액터가 된다. 그림 3 는 스마트 홈 시스템의 문맥 구조 모델을 보여준다.

■ **유즈케이스 모델링**

구조화된 문맥 개체와 요구사항으로부터 추출된 Use case 의 관계를 이용하여 Use case 다이어그램을 그린다. 이때, 문맥 개체가 액터가 되는데 stereotype 을 사용하여 액터의 카테고리를 표현한다. 그림 4 은 스마트 홈 시스템에 대한 유즈케이스 다이어그램을 보여준다.

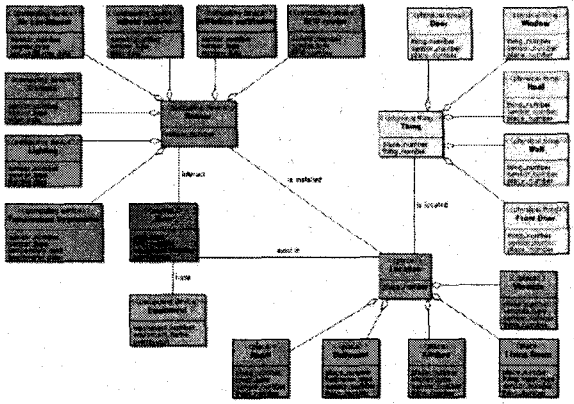


그림 3 스마트 홈 시스템의 문맥 구조 모델

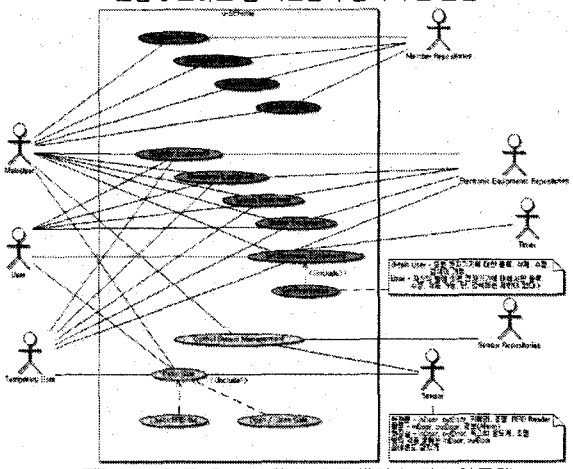


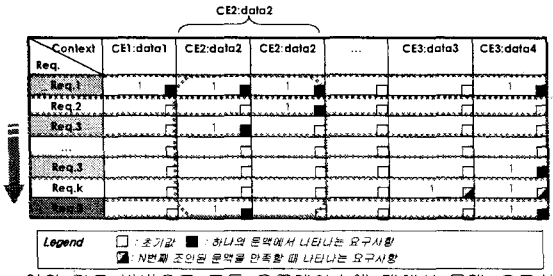
그림 4 스마트 홈 시스템의 유즈케이스 다이어그램

■ **문맥 기반 요구사항 일반화**

유즈케이스 다이어그램에 나타낸 각각의 유즈케이스에 대해서 소프트웨어 시스템을 블랙박스로 보고 소프트웨어와 문맥 개체간의 상호작용을 UML 의 시퀀스(Sequence) 다이어그램을 사용하여 분석한다. 이 다이어그램에 나타난 메시지의 흐름을 바탕으로 문맥과 문맥에 기반한 요구사항을 추출하여 문맥-요구사항 매트릭스를 생성한다. 문맥에서 요구사항이 나타나는 경우 문맥과 요구사항이 교차되는 칸에 숫자 1 을 기입한다. 매트릭스 행의 요구사항들은 문맥 개체가 일으킨 이벤트에 대응하여 시스템이 수행하는 오퍼레이션으로 구성된다. 하나의 문맥에 대해서 자기적응형 소프트웨어가 기능을 제공할 수도 있으나 두 개 이상의 문맥이 동시에 나타날 때 소프트웨어가 기능을 제공하는 경우도 있을 수 있다. 이를 구별하기 위해서 숫자를 기입하는 셀 내부에 오른쪽 아래 셀을 두고 하나의 문맥에서 나타나는 요구사항이면 셀을 ■표시하고, 두 개 이상의 문맥을 동시에 만족해야 하는 요구사항에 대해서는 해당 문맥과

요구사항이 교차하는 셀 내부에 ◪ 으로 표시한다. 이 때, 두 개 이상의 문맥이 동시에 나타나는 경우가 여러 개가 될 수 있으므로 이를 구별하기 위하여 작은 셀에 숫자를 기입한다.

표 3 문맥-요구사항 매트릭스



이와 같은 방법으로 모든 유즈케이스에 대해서 문맥-요구사항 매트릭스가 완성되면 문맥과 요구사항의 일반화 과정을 거친다. 문맥의 일반화는 반복되는 문맥을 하나로 합치면서 같은 문맥이나 흩어져 있던 요구사항을 하나의 문맥의 요구사항으로 통합하는 것을 말한다. 요구사항의 일반화 역시 반복되는 요구사항에 대해서 같은 작업을 하는 것이다. 문맥 기반 요구사항 일반화를 통해서 하나의 문맥에 어떠한 요구사항이 존재하는지, 하나의 요구사항이 어떠한 문맥에서 나타나는지를 분석해낼 수 있다.

■ 문맥에 따른 요구사항 가변점 식별

표 3 에서 Req.1 의 경우 CE1:data1, CE2:data2, CE3:data4 에 중복되어 나타난다. 이는 Req.1 이 여러 문맥에서 지속적으로 나타나는 공통적인 요구사항일 수도 있지만, Req.1 내부에 가변성이 숨겨져 있는 상태일 수 있다. 이를 다시 분석해서 각 문맥에서 똑같이 나타나는 것이 아니라 CE2:data2 문맥에서 원래 기능인 Req.1 와 차이가 나는 Req.1' 의 형태로 나타난다면 이는 문맥의 변화에 따라 다른 기능이 되는 것이므로 CE2:data2 문맥과 Req.1 이 교차하는 셀의 숫자를 2 로 기입한다. 이와 같이 요구사항의 가변성이 발견되면 기존의 요구사항과 차이를 보여주기 위하여 기존의 숫자에 1 을 증가한 숫자를 기입한다. 즉, 같은 행에서 셀의 숫자가 같으면 같은 기능이고, 숫자가 다르면 가변적인 기능이 된다. 표 4 는 문맥기반 요구사항 일반화 과정과 가변점 식별을 거친 스마트 홈 시스템의 문맥-요구사항 매트릭스이다. 요구사항 '*authenticate*' 는 *member repositories* 의 문맥개체를 이용한다. 그러나 이 때, 인식되는 문맥정보가 *family data* 인 경우와 *member data* 인 경우에 따라 이 요구사항의 기능이 가변성을 가지기 때문에 표 4 매트릭스에서 1 과 2 로 구분되어 표시되었다.

3.3 문맥 기반 아키텍처 설계 단계

문맥 기반 아키텍처 설계 단계에서는 자기적응형 소프트웨어가 동작 시에 문맥에 따라 적절한 서비스를 제공할 수 있도록 하기 위하여 앞 단계인 문맥 기반 요구사항 분석에서 추출된 문맥과 문맥 기반 요구사항을 바탕으로 문맥에 따라 컴포넌트를 추가, 삭제, 대체할 수 있는 자기적응형 소프트웨어 아키텍처를 생성한다.

표 4 일반화 과정과 가변점 식별을 거친 스마트 홈 시스템의 문맥-요구사항 매트릭스

entity: contextual data / requirement	member repositories: family data	member repositories: member data	Electronic Equipments Repositories equipments data	timer: time data	sensor: sensor data	sensor repositories: sensor data
authenticate	1◪	2◪				
connect	1◪	2◪	3◪	4◪	5◪	6◪
check user		1■				
display UI(family)	1■					
search family		1■				
register family		1■				
unregister family		1■				
update family		1■				
store family	1◪	2◪				
display UI(equipment)			1■			
search equipment			1■			
register equipment			1■			
unregister equipment			1■			
update equipment			1■			
use equipment			1◪	2◪		
store equipment			1■			
check permission			1■			
manage(equipment)			1◪	2◪		
display UI(sensor)					1■	
search sensor						1■
manage(sensor)						1■
check RFID						1■
check video data					1■	
deal with sensing data					1■	
open closeCommand					1■	

■ 서비스 패키지 추출

서비스 패키지란 기능적으로 관련 있는 액션들의 응집성 있는 집합이다. 이는 소프트웨어 동작시의 문맥에 따른 컴포넌트의 추가, 삭제, 대체를 위하여 문맥 개체가 발생시키는 하나의 이벤트에 대응하여 소프트웨어가 제공해야 하는 서비스에 나타나는 기능적으로 관련된 클래스들의 집합을 의미한다. 서비스 패키지를 추출하는 방법은 하나의 유즈케이스에 대하여 연관성 있는 문맥 개체들이 발생시키는 이벤트를 찾아서 각각의 이벤트에 대해서 UML 의 시퀀스 다이어그램을 이용하여 유즈케이스에 참여하는 클래스를 추출한다. 시퀀스 다이어그램에 나타난 클래스를 다음과 같은 기준으로 그룹핑함으로써 서비스 패키지를 추출한다.

- 문맥 개체가 발생시키는 문맥적 이벤트에 대해서 시퀀스 다이어그램에서 메시지의 호름이 이어지는 클래스들을 하나의 서비스 패키지로 식별한다.
- 문맥에 따라 선택적으로 나타나는 기능들은 개별적인 서비스 패키지로 식별한다.
- 추출된 서비스 패키지 내의 클래스들이 두 개 이상의 서비스 패키지에 걸쳐서 나타날 경우 이를 새로운 서비스 패키지로 식별한다.

■ 문맥 기반 구조 모델 생성

문맥 기반 구조 모델은 자기적응형 소프트웨어의 기능을 제공하는 모듈들의 구조화된 모습을 보여준다. 모델의 구성요소는 앞 단계에서 추출된 서비스 패키지와 이들 사이의 의존 관계, 관련된 문맥정보이다. 문맥에 따라 가변적으로 나타나는 서비스 패키지에 대해서는 스테레오 타입 <<variable>>을 사용하여 나타낸다. 그림 5 는 스마트 홈 시스템에 대한 문맥 기반 구조모델을 보여준다. 이 다이어그램에서 서비스 패키지에 영향을 주는 문맥정보는 노트 형태로 표현되어 해당 서비스 패키지와 연결되어 있다.

■ 컴포넌트 식별

문맥 기반 구조 모델의 서비스 패키지와 패키지 사이의 관계를 바탕으로 자기적응형 소프트웨어 동작 시의 변경 단위인 컴포넌트를 식별하고 컴포넌트의 인터페이스를 추출한다. 컴포넌트의 인터페이스란 컴포넌트들이 연결되기 위한

수단으로써 컴포넌트 사용자에 의해 호출되는 오퍼레이션들의 집합을 의미한다. 서비스 패키지와 서비스 패키지에서 제공하는 인터페이스를 식별하고 인터페이스를 구성하는 오퍼레이션들을 추출하고 나면 서비스 패키지가 컴포넌트로 매핑이 되고 오퍼레이션으로 구성된 인터페이스가 컴포넌트의 인터페이스가 된다.

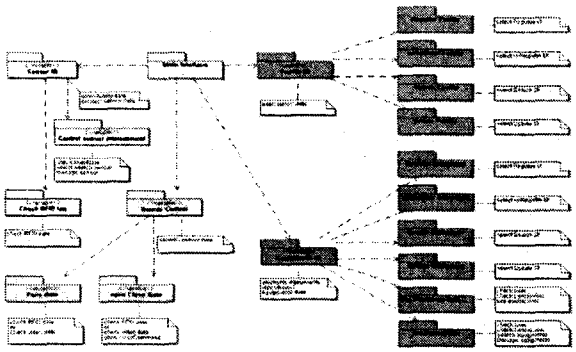


그림 5. 스마트 홈 시스템의 문맥 기반 구조 모델

■ **문맥 기반 행위 모델 생성**

문맥 기반 행위 모델을 구성하는 요소는 컴포넌트와 커넥터이다. 문맥 기반 행위 모델을 통해 자기적응형 소프트웨어 동작 시의 문맥의 변화에 따라 컴포넌트들이 커넥터를 통해 연결되는 동적인 모습을 볼 수 있다. 문맥 기반 행위 모델은 컴포넌트-커넥터 다이어그램과 컴포넌트 상호작용 다이어그램으로 구성된다. 커넥터는 자신에게 연결되어 있는 모든 컴포넌트에 대하여 소프트웨어 동작 시에 발생하는 문맥에 따라 어떠한 컴포넌트가 추가, 삭제, 대체되는 가에 대한 정보를 담고 있다. 동작 시의 문맥에 따라 컴포넌트와 커넥터가 연결되는 정보는 다음 세부 활동에서 커넥터 명세를 통해 명시적으로 표현한다. 컴포넌트 상호작용 다이어그램은 컴포넌트 커넥터 다이어그램에 나타난 컴포넌트들이 커넥터를 통해 어떠한 메시지를 교환하는가를 표현한다.

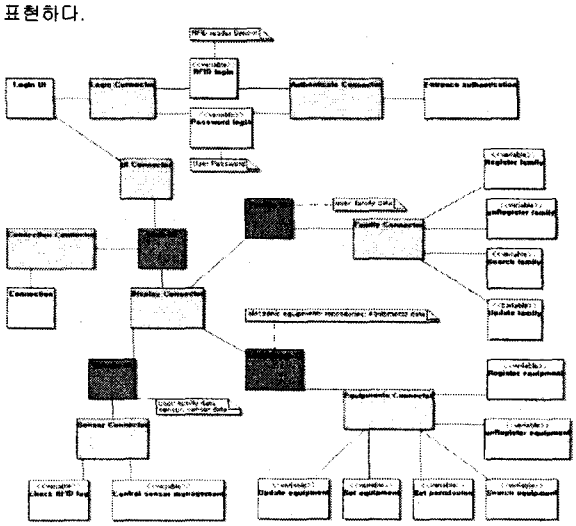


그림 6. 스마트 홈 시스템의 컴포넌트-커넥터 다이어그램

그림 6 은 스마트 홈 시스템에 대한 컴포넌트-커넥터 다이어그램이다. 이 단계에서 *authenticate* 요구사항은 *Login UI, RFID login, Password login, Entrance authentication* 컴포넌트 추출되었다. 이 중 각기 다른 문맥정보인 *family data* 와 *member data* 의 처리를 위하여 *RFID login, Password login* 컴포넌트가 가변적 컴포넌트로 구분되어 모델링되었다. 그림 7 은 *Family UI* 컴포넌트의 상호작용 다이어그램이다.

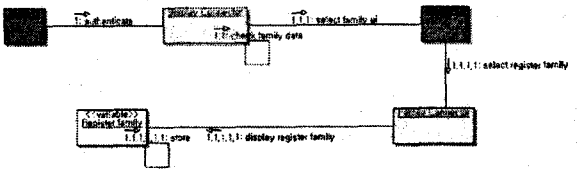


그림 7. *Family UI* 의 컴포넌트 상호작용 다이어그램

문맥의 변화에 따른 컴포넌트의 변경에 대한 정보를 제공하고 이를 관리하기 위하여 문맥-컴포넌트 매트릭스를 생성한다. 해당 문맥에서 컴포넌트의 추가, 삭제가 발생할 경우 문맥과 컴포넌트가 교차하는 셀에 Add, Delete 를 기입한다. 문맥 기반 요구사항 분석 단계에서 추출된 문맥과 문맥 기반 요구사항들이 문맥 기반 아키텍처 설계 과정을 거치면서 문맥과 문맥에 따라 변화하는 컴포넌트로 매핑이 되는 것이다. 문맥-컴포넌트 매트릭스를 통해 하나의 문맥에서 어떤 컴포넌트들이 추가되고 삭제되는지, 하나의 컴포넌트가 어떤 문맥에서 추가되고 어떤 문맥에서 삭제되는지에 대한 정보를 얻을 수 있다. 표 5 는 스마트 홈 시스템에 대한 문맥-컴포넌트 매트릭스를 보여준다.

표 5 문맥-컴포넌트 매트릭스

context / component	member repositories: member data	Electronic Equipments Repositories: equipments data	sensor: sensor data	sensor repositories: sensor data
Login UI	Add, Delete	Add, Delete	Add, Delete	Add, Delete
RFID login	Add, Delete	Add, Delete	Add, Delete	Add, Delete
Password login	Add, Delete	Add, Delete	Add, Delete	Add, Delete
Entrance authentication				
Main UI	Add			
Family UI	Add			
Register family	Add, Delete			
Unregister family	Add, Delete			
Search family	Add, Delete			
Update family	Add, Delete			
Equipments UI			Delete	
Register equipment		Add, Delete		
unRegister equipment		Add, Delete		
Search equipment		Add, Delete		
Update equipment		Add, Delete		
Set equipment		Add, Delete		
Set permission		Add, Delete		
Sensor UI			Add, Delete	
Check RFID tag			Add, Delete	
Central sensor management			Add, Delete	

■ **컴포넌트와 커넥터 명세화**

문맥 기반 행위 모델을 생성하면 문맥 기반 행위 모델을 구성하는 컴포넌트와 커넥터의 자세한 정보를 제공할 수 있는 컴포넌트와 커넥터의 명세를 기술한다. 표 6 은 *Family UI* 에 대한 컴포넌트 명세이다.

표 6 *Family UI* 에 대한 컴포넌트 명세

Component name	Family UI		
Purpose	가족구성원을 등록/삭제/검색/수정을 제프롭한다.		
Interface	Interface name	IselectFamily UI	
	Purpose	Main UI에서 user가 select family UI를 선택했을 때 발생하는 발생한다.	
	Operation	Operation name	select family UI
		Constraint	mainUser만 접근할 수 있다.
		input	
		output	
		Precondition	mainUser가 인증과정을 거친 상태이다.
		Postcondition	Family UI가 활성화된다.
Required Interface	Interface name	IFamilyConnection	
	Purpose	가족구성원을 등록/삭제/검색/수정할 수 있는 화면을 연결한다.	
	Operation	Operation name	display register family() display unregister family() display search family() display update family()
		Constraints	
		input	
		output	connected
		Precondition	mainUser가 인증과정을 거친 상태이다.
		Postcondition	Family UI가 활성화된다.

자기적응형 소프트웨어 설계와 기존의 다른 소프트웨어 설계의 큰 차이는 커넥터이다. 자기적응형 소프트웨어는 문맥에 따라 다른 컴포넌트가 추가, 삭제, 대체가 가능하기 때문에 컴포넌트 사이의 연결을 나타내는 커넥터에도 이러한 자기적응형 소프트웨어의 특징이 반영되어야 한다. 즉 문맥에 따라 다른 기능을 제공하는 컴포넌트가 있을 때, 커넥터는 자신에게 연결되어 있는 컴포넌트를 식별하고 문맥의 변화가 발생했을 때 기존의 컴포넌트와의 연결을 끊고 새로운 컴포넌트와 연결을 해야 한다. 이러한 정보를 반영하여 커넥터 명세를 기술한다. 커넥터 명세에서의 역할과 결합의 프로토콜은 CSP 의 일부를 사용하여 표현한다. 표 7 은 *Family connector* 에 대한 커넥터 명세이다.

표 7 커넥터 명세

Connector name	Family connector	
Role	Role name	select register family
	Description	select?main User_data□
	Role name	display register family
	Description	display?family_data□
Glue	Contextual condition	register family data
	Description	Register Family.display?register data□
	Contextual condition	register family data
	Description	Register Family.store?register data□

CSP Notation

e?x : input of data　　e!x : ouput of data

e->P : e then P　　√ : successful termination

P□Q : P or Q, deterministic choice by the environment

P ⫫ Q : P or Q, non deterministic choice by the process itself

4. 결론 및 향후 연구

본 논문에서는 소프트웨어 설계 방법의 전 과정에 걸쳐 문맥의 변화에 따른 자기적응형 소프트웨어의 행위의 변화를 분석하고 설계할 수 있는 자기적응형 소프트웨어 설계 방법을 제시하였다. 이 방법을 통해 요구사항으로부터 자기적응형 소프트웨어가 동작하는 외부 환경을 나타내는 문맥과 기능을 추출하고 이를 분석하여 동적으로 적응 가능한 아키텍처와 아키텍처를 구성하는 컴포넌트와 커넥터를 설계 할 수 있도록 하였다. 또한 이 방법을 스마트 홈 시스템 개발에 적용해 봄으로써 방법의 적용가능성을 검증해 보았다. 그 결과 이 방법을 통해 소프트웨어 개발자에게 자기적응형 소프트웨어가 동작하는 환경인 문맥을 분석한 정보와 함께 문맥의 변화에 따라 재구성되는 컴포넌트와 커넥터 명세를 제공함으로써 자기적응형 소프트웨어 개발을 효율적으로 지원할 수 있음을 확인하였다.

현재 이 연구의 확장으로 스마트 홈 시스템의 개발을 좀 더 다양한 형태의 문맥정보에서 검증하기 위한 도구로서 문맥인식(context-aware) 홈을 위한 시뮬레이터를 개발 중에 있다. 향후에는 이를 이용하여 본 논문에서 제시한 자기적응형 소프트웨어 설계 방법을 소프트웨어의 비기능적 요구사항에 대해서도 적용할 수 있도록 확장할 수 있을 것이다.

5. 참고문헌

[1] 유승화, *유비쿼터스 사회의 RFID*, 전자신문사, 2005.

[2] Self adaptive software, DARPA, BAA 98-12, Proposer Information Pamphlet, 1997.

[3] Anand Ranganathan, Roy H. Campbell, "An infrastructure for context-awareness based on first order logic", Personal and Ubiquitous Computing,Vol. 7 no.6, pp.353-364, 2003.

[4] Anind K. Dey, Daniel Salber and Gregory D. Abowd, " A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications" , Anchor article of a special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal, Vol. 16 No.2-4, pp. 97-166, 2001.

[5] Daniel Salber, Anind K. Dey and Gregory D. Abowd, " The Context Toolkit: Aiding the Development of Context-Enabled Applications" , Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, pp. 434-441, 1999.

[6] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Modeling context information in pervasive computing systems", Proceedings of the 1st International Conference on Pervasive Computing, Zurich, Switzerland, Springer-Verlag, pp.167-180, 2002.

[7] Robert Laddaga, Paul Robertson, Howard E. Shrobe, "Probabilistic Dispatch, Dynamic Domain Architecture, and Self-Adaptive Software", Self-Adaptive Software, Second International Workshop, IWSAS 2001, LNCS 2614, Springer-Verlag, pp.227-237, 2003.

[8] David Garlan, and Bradley Schmerl, "Model-based Adaptation for Self-Healing Systems", Proceedings of the first workshop on Self-healing systems 2002, pp.27-32, 2002.

[9] Shang-Wen Cheng, An-Cheng Huang, David Garlan, Bradley Schmerl, and Peter Steenkiste, "Rainbow: Architecture-Bases Self Adaptation with Reusable Infrastructure," IEEE Computer Vol. 37 No. 10, 2004.