적응형 소프트웨어 개발을 위한 문맥 기반 요구사항 분석 방법

장호진[0], 문미경, 염근혁
부산대학교 컴퓨터 공학과
{hojin[0], mkmoon, yeom}@pusan.ac.kr

An Approach to Context-based Requirement Analysis for Self-Adaptive Software Development

Hojin Jang[0], Mikyeong Moon, Keunhyuk Yeom
Dept. of Computer Engineering, Pusan National University

요 약

소프트웨어의 외부 환경이 동적으로 변화하고 복잡해지면서 소프트웨어가 예상하지 못한 외부 환경의 변화에 직면하였을 때 변화를 감지하고 대안을 선택하여 지속적인 서비스를 제공할 필요성이 증가하고 있다. 이를 위해 외부 환경의 변화를 감지하고 변화에 적응할 수 있는 적응형 소프트웨어가 나오게 되었다. 그러나 적응형 소프트웨어를 개발하고자 할 때 기존의 요구사항 분석 방법은 소프트웨어의 외부 환경의 변화에 대한 고려가 부족하다. 본 논문에서는 적응형 소프트웨어의 외부 환경의 변화와 그러한 변화에 의해 가변적으로 나타나는 요구사항을 분석하기 위한 문맥 기반 요구사항 분석 방법을 제시한다.

1. 서 론

오늘날의 소프트웨어가 직면하고 있는 문제 중 한 가지는 소프트웨어가 동작하는 환경이 잘 정의되고 고정되어 있는 것이 아니라 동적으로 변화함에 따라 복잡해지고 있다는 것이다. 따라서 소프트웨어가 예상하지 못한 외부 환경의 변화에 직면하였을 때 동작을 멈추는 것이 아니라, 변화를 감지하고 대안을 선택하여 지속적으로 서비스를 제공할 필요가 있다. 이에 대한 해결 방법으로 적응형 소프트웨어가 제안되었다. 적응형 소프트웨어란 자신의 행위를 평가하여 소프트웨어가 원래 의도한 것을 수행하지 못하거나, 더 나은 기능을 수행하는 것이 가능하다면 스스로 행위를 변경할 수 있는 소프트웨어를 말한다[1]. 이러한 적응형 소프트웨어를 개발하기 위해서는 소프트웨어의 기능을 분석하고 설계하는 것 뿐 아니라 소프트웨어가 동작하는 외부 환경의 변화를 분석하고 설계하는 것이 병행되어야 한다. 그러나 현재 적응형 소프트웨어에 대한 연구[2][3]는 소프트웨어가 외부 환경의 변화에 '어떻게' 적응하도록 할 것인가에 대한 적응 메카니즘에 초점이 맞추어져 있어 적응형 소프트웨어 외부 환경을 모델링하는 기법이나 적응을 위해 대체할 수 있는 기능들을 찾아내는 방법에 대한 연구가 부족하다. 따라서 본 논문에서는 적응형 소프트웨어의 외부 환경의 변화와 변화에 의해 가변적으로 나타나는 요구사항을 분석하기 위한 적응형 소프트웨어의 문맥 기반 요구사항 분석 방법을 제시한다.

* 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10197-0)지원으로 수행되었음

2. 관련 연구

2.1. MIT의 DDA(Dynamic Domain Architecture)

DDA[4]는 어플리케이션을 공통적인 서비스들의 계층(Layers of common services)으로 구조화하는데, 공통적인 서비스 계층은 상이한 외부 환경의 조건에 따른 가변적인 구현(variant implementation)들로 구성된다. 이 때 목적 링크(purpose links)가 있어서 외부 환경의 조건에 따라 가변적인 구현들 중에서 적절한 구현을 선택할 수 있는 정보를 제공한다. DDA는 각 컴포넌트에 대해 대체될 수 있는 구현(alternative implementation)을 가지고 있어서 현재 어플리케이션의 동작이 실패했을 때 대체될 수 있는 구현을 호출하고 실패에서 회복하게 된다. DDA의 연구를 잘 살펴보면 도메인 공학에서 도메인 내의 여러 시스템의 공통적인 기능(common functionality)을 찾아내고 공통적인 기능 속에서 가변성(variability within commonality)을 분석하는 것이 적응형 소프트웨어의 공통 서비스 계층의 대체될 수 있는 구현을 찾아내는 것과 동일함을 알 수 있다.

2.2. 적응형 소프트웨어 개발에서의 도메인 분석 방법

도메인 분석 방법은 C&V(Commonality and Variability)를 추출하는데 초점이 있다. 그러나 이것은 C&V를 이용하여 관련된 여러 시스템을 효과적으로 개발하려는 목적을 가지고 있는데 반해, 적응형 소프트웨어에서는 적응형 소프트웨어 하나를 개발하기 위한 목적으로 C&V를 이용한다. 그림 1은 도메인 분석 방법에서의 C&V의 이용과 적응형 소프트웨어에서의 C&V이용의 차이점을 보여준다. 그림 1의 왼쪽 아래 부분은 개발하려는 시스템의 특징에 맞게 각 패키지 내에 있는 서비스들 중 0개 이상씩(0개인 경우는 선택적 기능을 의미)을

선택하여 관련된 유사한 시스템들을 개발하는 모습을 나타낸다. 그러나 그림 1의 오른쪽 위 부분은 적응형 소프트웨어 시스템의 개발 모습을 나타내고 있는데, 도메인 안의 유사 시스템과는 달리 모든 가변성을 다 포함한 패키지 형태의 기능들을 포함하여 시스템을 개발하는 것을 보여주고 있다.

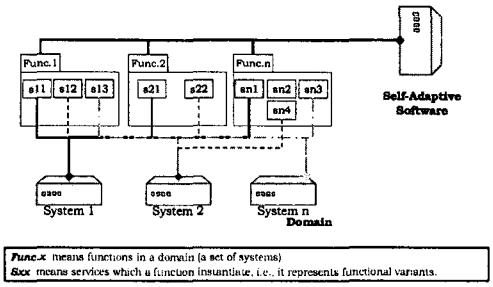

그림 1 도메인 분석 방법과 적응형 소프트웨어

3. 문맥 기반 요구사항 분석 방법

본 논문에서 제시하는 적응형 소프트웨어를 위한 문맥 기반 요구사항 분석 방법에서는 문맥 기반 요구사항 분석을 위한 Scoping을 한 후, 문맥 기반 요구사항 분석을 진행한다. 그림 2는 문맥 기반 요구사항 분석 방법의 과정을 보여준다.

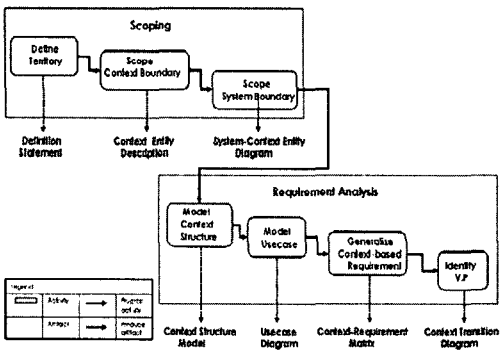

그림 2 문맥 기반 요구사항 분석 방법의 전체 과정

3.1 Scoping

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

3.1.1 영역 정의 (Define Territory)

문맥 정의에서는 개발하고자 하는 소프트웨어와 소프트웨어가 동작하는 환경인 문맥을 자연어로 기술한다. 또한 문맥의 기술적인 제약사항, 서비스 제약사항 등을 같이 기술한다.

3.1.2 문맥 범위 결정 (Scope Context Boundary)

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. 문맥을 분석하여 카테고리에 해당되는 문맥 개체들을 추출하여 표 1과 같이 문맥 개체 서술(Context Entity Description)을 작성한다.

표 1 Context Entity Description

Category		Name	Description
Place			
Person			
Thing	Physical Thing		
	Computing Device		

3.1.3 시스템 범위 결정 (Scope System Boundary)

소프트웨어는 외부 환경에 존재하는 문맥 개체와 상호작용을 한다. 문맥 범위 결정에서 추출된 문맥 개체와 소프트웨어의 관계를 모델링함으로써 소프트웨어의 시스템 범위를 결정한다. 소프트웨어와 문맥 개체 사이의 상호작용은 그림 3과 같은 시스템-문맥 개체 다이어그램(System-Context Entity Diagram)으로 표현한다.

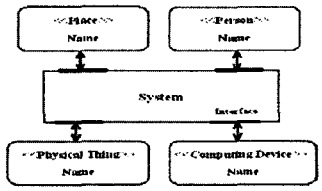

그림 3 System-Context Entity Diagram

3.2 문맥 기반 요구사항 분석

문맥 기반 요구사항 분석에서는 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 상세화된다.

3.2.1 문맥 구조 모델링 (Model Context Structure)

UML의 클래스 다이어그램을 이용하여 Scoping 단계에서 추출된 문맥 개체들을 구조화한다. 문맥 개체의 카테고리는 stereotype으로 표현한다. 이렇게 구조화된 문맥 개체들은 유즈케이스 다이어그램에서 액터가 된다. 그림 4는 문맥 구조 모델(Context Structure Model)의 Meta-model을 보여준다.

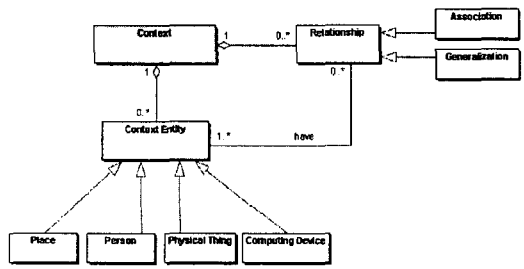

그림 4 Context Structure Model의 Meta-model

3.2.2 유즈케이스 모델링 (Model Usecase)

이전 단계에서 구조화된 문맥 개체와 요구사항으로부터 추출된 유즈케이스의 관계를 이용하여 유즈케이스 다이어그램을 그린다. 이 때, 문맥 개체가 액터가 되는데 stereotype을 사용하여 액터의 카테고리를 표현한다.

3.2.3 문맥 기반 요구사항 일반화 (Generalize Context-based Requirement)

유즈케이스에 대해서 소프트웨어와 문맥 개체간의 상호작용을 분석하여 "문맥 C1에서 기능 Func.1을 수행한다"의 문장 형식으로 문맥과 문맥에 기반한 요구사항을 추출한다. 문장에 나타난 문맥과 기능을 그림 5와 같이 매트릭스 형태로 표현한다. 매트릭스의 열에는 문맥을, 행에는 요구사항을 기입하고, 문맥에서 요구사항이 나타나는 경우 문맥과 요구사항이 교차되는 칸에 O를 표시한다. 이와 같은 방법으로 모든 유즈케이스에 대해서 매트릭스가 완성되면 문맥과 요구사항의 일반화과정을 거친다. 문맥의 일반화는 중복된 문맥을 하나로 합치면서 같은 문맥이나 흩어져 있던 요구사항을 하나의 문맥의 요구사항으로 통합하는 것을 말한다. 요구사항의 일반화 역시 중복된 요구사항에 대해서 같은 작업을 하는 것이다. 문맥 기반 요구사항 일반화를 통해 하나의 문맥에 어떠한 요구사항이 존재하는지, 하나의 요구사항이 여러 문맥에서 나타나는지를 분석해낼 수 있다. 예를 들어 그림 5에서 문맥 Context 2에서는 요구사항 Req.1, Req.3에 해당하는 기능이, 문맥 Context 3에서는 요구사항 Req.1, Req.2에 해당하는 기능이 수행된다. Context 2와 Context 3이 하나의 문맥으로 일반화되면서 요구사항들이 합쳐지게 된다. 즉, 일반화된 문맥 Context m에서는 요구사항 Req.1, Req.2, Req.3이 나타나는 것이다.

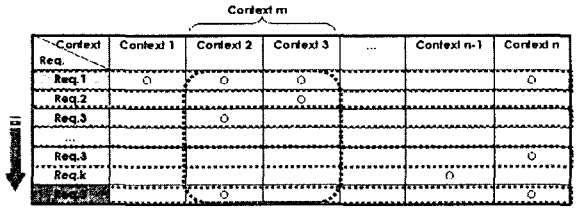

그림 5 Context-Requirement Matrix

3.2.4. 문맥에 따른 요구사항 가변점 식별 (Identify V.P)

그림 5에서 Req.1의 경우 Context 1, Context m, Context n에 중복되어 나타난다. 이는 Req.1이 여러 문맥에서 지속적으로 나타나는 공통적인 요구사항일 수도 있지만, Req.1 내부에 가변성이 숨겨져 있는 상태일 수 있다. 이를 분석하기 위하여 문맥 전이 다이어그램(Context Transition Diagram)을 이용한다. 먼저, 1)두 개 이상의 문맥에 공통적으로 나타나는 요구사항이 있을 때 문맥들을 그림 6과 같은 문맥 전이 다이어그램으로 표현한다. 2)문맥 전이 다이어그램에 나타난 문맥의 흐름을 따라서 Req.1을 다시 분석해보고 각 문맥에서 똑같이 나타나는 것이 아니라 Context m에서 원래 기능인 Req.1와 차이가 나는 Req.1'의 형태로 나타난다면 이는 문맥에 따라 다른 기능이 되는 것이므로 <<V.P>>라고 표시한다.

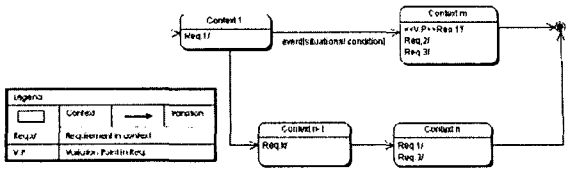

그림 6 Context Transition Diagram

이러한 과정을 통해 분석된 3)문맥 기반 요구사항의 가변 정보는 표 2와 같은 형태로 기술된다.

표 2 문맥 기반 요구사항 가변성 정보

Variation Point name	Requirement name	Context name	Description
$V.P_1$	Req.1	Context m	different UI
...	...	...	...

4. 결론 및 향후 연구

본 논문에서는 적응형 소프트웨어의 요구사항이 문맥에 따라 다르게 나타나는 것을 인식하고 적응형 소프트웨어를 개발하기 위하여 문맥 기반 요구사항을 분석하는 방법을 제시하였다. 본 논문에서 제시한 분석 방법의 핵심은 Context-Requirement Matrix이다. 이 매트릭스에는 문맥과 문맥에 기반한 요구사항의 관계가 잘 분석되어 있다. 이 매트릭스를 설계 단계에서 이용한다면 컴포넌트 추출과 아키텍처 개발에 문맥과 문맥에 기반한 요구사항이 반영되어 문맥의 변화에 적절한 서비스를 제공할 수 있는 소프트웨어를 개발할 수 있게 된다. 향후 연구로는 현재의 문맥 기반 요구사항 분석을 바탕으로 한 적응형 소프트웨어 설계, 구현에 대한 연구가 필요하다.

5.참고문헌

[1] Self adaptive software, DARPA, BAA 98-12, Proposer Information Pamphlet, 1997. URL:http://www.darpa.mil/ito/Solicitations/PIP_9812.html.

[2] Oreizy, P., Gorlick, M.M., Taylor R.,N., Johnson, G., Medvidovic, N., Quilici, A., Rosen-blum, D., and Wolf, A., "An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, 14(3) pp.54-62, 1999.

[3] 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.

[4] Robert Laddaga, Paul Robertson, Howard E. Shrobe, "Probabilistic Dispatch, Dynamic Domain Architecture, and Self-Adaptive Software", Robert Laddaga, Paul Robertson, Howard E. Shrobe editors Self-Adaptive Software, Lecture Notes in Computer Science 2614 Springer, pp.227-237, 2003.