적응형 소프트웨어 아키텍처를 위한 휘쳐 기반의 상황 모델링 기법

서만수[O] 박수용 조훈
서강대학교 소프트웨어공학 연구실
국방과학연구소
msseo@selab.sogang.ac.kr[O], sypark@ccs.sogang.ac.kr

Feature-Based Situation Modeling Technique for Adaptive Software Architecture

Mansoo Seo[O], Sooyong Park, Hun Cho
Dept. of Computer Science & Engineering, Sogang University
Agency for Defense Development

요 약

우리의 일상생활에서 차지하고 있는 소프트웨어의 비중은 점점 더 커져가고 있으며, 그만큼 소프트웨어의 오류가 우리에게 미치는 영향도 커지고 있다. 또한, 현대의 소프트웨어는 더 복잡해지고 높은 안정성이 필요하게 되므로 기존 소프트웨어와는 다른 새로운 패러다임의 소프트웨어가 필요하다. 적응형 소프트웨어는 소프트웨어 스스로가 자신의 기능을 추론하고 오류가 발생하면 이에 적절하게 대처할 수 있는 새로운 소프트웨어 패러다임이다. 적응형 소프트웨어를 개발하려면 소프트웨어가 적응해야 하는 상황을 모델링 하는 기술이 필요하다. 본 논문에서는 프로덕트 라인 개발 방법에서 사용하는 휘쳐 모델을 사용해 적응형 소프트웨어의 상황을 모델링 하는 기법을 제안한다.

1. 서 론

오늘날의 소프트웨어 공학이 직면하고 있는 어려움은 크게 소프트웨어의 복잡도 증가, 소프트웨어의 견고성 부족, 소프트웨어의 독립성 보장 요구 이 세가지로 분류할 수 있다[1]. 이 세가지 문제점은 따로 해결할 수 있는 것이 아니라 서로 연관되어 소프트웨어 개발을 더욱 어렵게 하고 있다. 예를 들어, 실시간 내장형 시스템의 경우 외부의 간섭 없이 동작하도록 소프트웨어의 독립성을 보장해야 하는데, 이런 경우에는 높은 견고성을 요구하며, 그에 따라 소프트웨어의 복잡도는 더욱 증가하게 된다.

또한, 우리의 일상생활에서 차지하는 소프트웨어의 비중은 점점 더 커지고 있으며, 그만큼 소프트웨어의 오류가 우리에게 미치는 영향 또한 커지고 있다. 이처럼 높은 안정성을 요구하는 시스템을 오류 없이 수행시키려면 기존의 소프트웨어와는 다른 새로운 패러다임의 소프트웨어가 필요하다.

적응형 소프트웨어란 자신의 행위를 평가해서 원래 의도했던 것을 수행하지 못하고 있거나, 또는 더 나은 기능을 수행할 수 있다고 예상되면 스스로 행위를 변경할 수 있는 소프트웨어이다[2]. 소프트웨어가 실행을 중단하지 않고 변경이 되려면 아키텍처에 기반한 동적 적응 방법이 필요하다. 아키텍처에 기반한 적응형 소프트웨어 개발 연구[3][4]는 이미 여러 방면에서 수행되고 있지만, 적응형 소프트웨어 개발의 사전 단계로서, 소프트웨어가 적응해야 하는 상황을 모델링 하는 연구는 거의 전무한 실정이다.

* 본 연구는 국방과학연구소 위탁연구 No. UD020021FD 지원에 의한 것임.

본 논문에서는 프로덕트 라인 개발방법에서 사용하는 휘쳐 모델을 기반으로 해서 소프트웨어가 적응해야 하는 상황을 모델링 하는 기법을 제안한다.

2장에서는 소프트웨어가 적응해야 하는 상황을 모델링 하는 본 연구의 배경을 기술하며, 3장에서는 관련 연구인 아키텍처 기반의 적응형 소프트웨어와 FORM[5]을 소개한다. 4장에서는 휘쳐 모델을 이용하여 적응형 소프트웨어의 상황 모델링 하는 방법을 제안하고, 5장에서는 결론 및 향후 연구에 대해 기술한다.

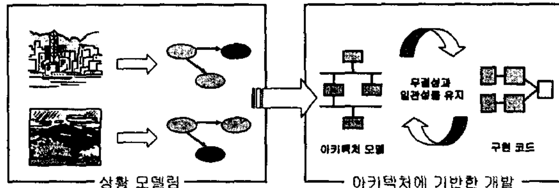
2. 연구 배경

적응형 소프트웨어를 개발하기 위해서는 형식화 언어, 프로그래밍 언어, 객체지향 분석 방법 등 여러 가지 관점에서 접근이 가능하다. 그러나, 소프트웨어의 변경에 대한 계획, 조정, 관찰, 평가, 적응 등을 통합적인 관점에서 관리하려면 소프트웨어 아키텍처를 중심으로 한 적응이 이루어져야 한다. 즉, 소프트웨어의 전체적인 구성, 제약 사항 등을 표현하는 아키텍처에 기반하여 외부 환경을 관찰하고 평가하여 적절한 적응을 해야 한다.

아키텍처에 기반한 적응형 소프트웨어 접근 방법은 외부 환경을 관찰, 평가한 후 적절한 계획을 세우는 적응 관리(Adaptation Management)와 계획에 의하여 아키텍처 모델과 수행 중인 코드가 동적으로 적응하는 변경 관리(Evolution Management)의 두 단계로 나누어진다[3]. 하지만, 적응형 소프트웨어를 실제적으로 개발하기 위한 방안을 제시하지는 않는다.

본 논문에서는 적응형 소프트웨어 개발을 위해 [그림 1]과 같이 소프트웨어가 적응해야 하는 상황을 모델링 하는 부분과 아키텍처에 기반한 코드 작성을 하는 두 부분으로 나누는 것을 제안한다. 그리고, 프로덕트 라인

개발 방법에서 사용하는 휘쳐 모델을 사용하여 적응형 소프트웨어의 상황 모델링을 보다 체계적으로 하는 방법을 제안한다.



[그림 1] 적응형 소프트웨어 개발을 위한 접근 방법

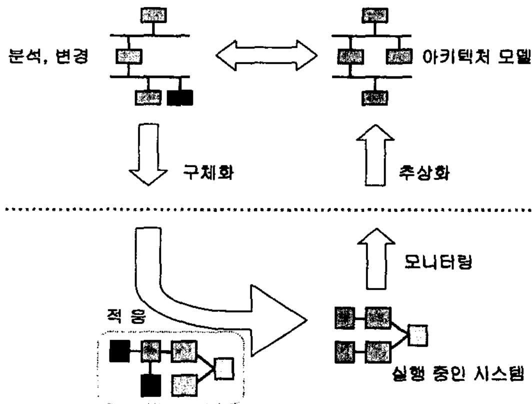
3. 관련 연구

3.1 아키텍처 기반의 적응형 소프트웨어

적응형 소프트웨어에 대한 접근 방법은 여러 가지가 있다. 프로그래밍 언어 수준에서의 예외 처리나 동적 알고리즘의 교체 등 다양한 수준에서 소프트웨어의 적응을 다룰 수 있다. 그러나 이러한 방법들은 시스템 전체의 적응을 다루기에는 적절하지 못하며, 적응 전략을 바꾸는데도 유연성이 떨어진다.

소프트웨어의 적응을 소프트웨어의 추상화 모델인 아키텍처 관점에서 다루면 이러한 문제점들을 해결할 수 있다. [그림 2]는 아키텍처 기반의 적응 메커니즘을 나타낸 것이다[4]. 시스템의 실행 상태를 외부에 있는 컴포넌트가 모니터링하고 문제점이 있을 경우, 이를 아키텍처 모델로 추상화시킨다. 아키텍처 수준에서 문제점을 분석, 변경 한 후 다시 이를 구체화시켜 실행 중인 시스템에 적응시킨다.

아키텍처에 기반한 적응형 소프트웨어는 적응 메커니즘이 어플리케이션에 독립적이므로 쉽게 모델들을 변경할 수 있으며 재사용성도 높다.



[그림2] 아키텍처에 기반한 적응

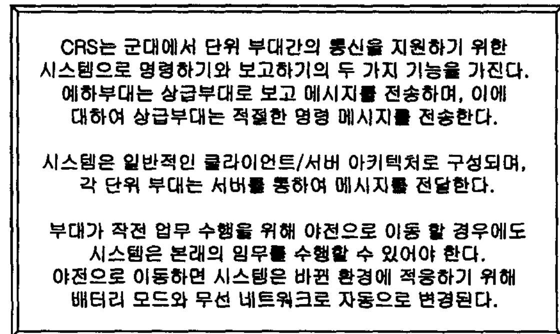
3.2 FORM (Feature-Oriented Reuse Method)

FORM[5]은 요구공학을 위해 휘쳐 모델을 사용하는 FODA (Feature-Oriented Domain Analysis)[6]를 설계와 구현 단계까지 확장한 개발 방법으로 도메인 내에서 어플리케이션의 공통점과 차이점을 휘쳐의 관점에서 찾아내고, 분석된 결과를 바탕으로 도메인 아키텍처와 컴포넌트를 개발하는 체계적인 소프트웨어 개발 방법이다.

도메인이 공통점과 차이점을 가진 연산 단위로써 기술되고 설명된다면, 재사용 가능한 다양한 아키텍처의 구성을 만들어 여러 어플리케이션을 개발할 수 있게 된다.

4. 적응형 소프트웨어 아키텍처를 위한 상황 모델링 기법

본 장에서는 적응형 소프트웨어 아키텍처를 위한 상황 모델링 기법을 기술한다. 제안하는 모델링 기법은 FORM[5]의 도메인 공학 부분 절차를 따라 크게 문맥 분석, 휘쳐 분석, 참조 아키텍처 설계의 세 단계로 이루어지며 CRS(Command and Report System)라는 예제를 적용해 기술한다. [그림 3]은 CRS의 요구사항을 나타낸 문서이다.


CRS는 군대에서 단위 부대간의 통신을 지원하기 위한 시스템으로 명령하기와 보고하기의 두 가지 기능을 가진다. 예하부대는 상급부대로 보고 메시지를 전송하며, 이에 대하여 상급부대는 적절한 명령 메시지를 전송한다.

시스템은 일반적인 클라이언트/서버 아키텍처로 구성되며, 각 단위 부대는 서버를 통하여 메시지를 전달한다.

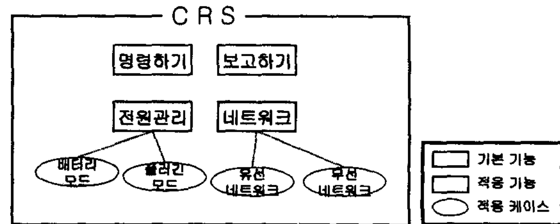
부대가 작전 업무 수행을 위해 야전으로 이동 할 경우에도 시스템은 본래의 임무를 수행할 수 있어야 한다. 야전으로 이동하면 시스템은 바뀐 환경에 적응하기 위해 배터리 모드와 무선 네트워크로 자동으로 변경된다.

[그림 3] CRS의 문제 영역 기술서

4.1 문맥 분석

문맥 분석 단계에서는 사용자의 요구사항을 기반으로 하여 개발하려는 적응형 소프트웨어의 적응 범위를 설정한다. 적응형 소프트웨어를 크게 기본 기능과 적응 기능으로 구분하는데, 적응 기능은 다시 적응되는 케이스에 따라 여러 가지로 분류된다.

CRS는 부대간의 원격통신을 위해 기본적으로 명령하기와 보고하기의 2가지 기능을 가진다. 또한, 야전환경에서도 임무 수행을 위해 전원과 네트워크 환경을 변경할 수 있다. [그림 4]는 CRS의 문맥 다이어그램으로 CRS의 적응범위를 나타낸다.



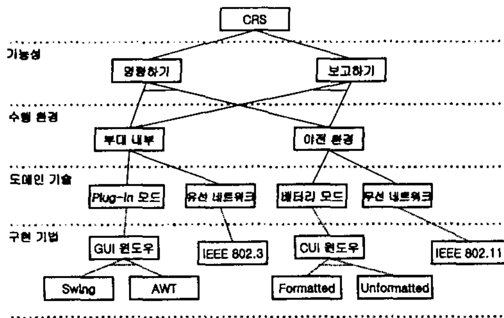
[그림 4] CRS의 문맥 분석 다이어그램

4.2 휘쳐 분석

휘쳐란 시스템에서 식별 가능한 추상화의 단위로써, 고객과 개발자가 종종 제품의 특성을 제품이 가지고 있는 특징(feature)으로 대화하는 것으로부터 유래되었다[6].

적응형 소프트웨어 개발에서 휘쳐 분석은 소프트웨어의 적응 범위 내에서 기본 기능과 적응 기능을 구분하여 휘쳐 모델을 생성하는 단계이다.

CRS는 명령하기와 보고하기의 기능을 가지며, 부대 내부에서는 유선 네트워크와 GUI기반의 전원 모드를 사용한다. 야전환경에서는 무선 네트워크와 배터리를 사용하는데, 전원 절약을 위해 명령어 기반의 인터페이스를 사용한다. [그림 5]는 문맥 분석 다이어그램을 바탕으로 작성한 CRS의 휘쳐모델이다.

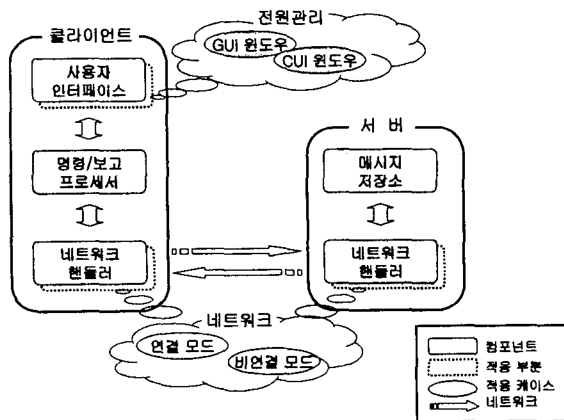


[그림 5] CRS의 휘쳐 모델

4.3 참조 아키텍처 설계

참조 아키텍처란 도메인 공학에서 선택된 휘쳐를 기반으로 설계된 아키텍처로서, 어플리케이션 공학 에서 재사용이 가능한 아키텍처의 템플릿을 말한다[5].

적응형 소프트웨어 개발에서 참조 아키텍처는 기본 기능을 중점으로 해서 설계되고, 적응되는 각 상황 별로 여러 개의 아키텍처로 구분된다. [그림 6]은 CRS의 참조 아키텍처로 사용자 인터페이스 부분과 네트워크 핸들러 부분이 CRS의 적응 가능한 부분임을 나타낸다.



[그림 6] CRS의 참조 아키텍처

5. 결론 및 향후 연구

앞으로 소프트웨어는 점점 더 복잡해지고 보다 높은 견고성과 독립성을 요구하게 된다. 또한, 소프트웨어가 우리의 실생활에서 차지하는 비중이 점점 커지게 되므로 보다 안정적인 소프트웨어의 개발이 절실하게 되었다. 적응형 소프트웨어는 자신의 행위를 추론하여 오류가 발생하면 스스로 이에 적절하게 대처할 수 있는 새로운 패러다임의 소프트웨어이다.

본 논문에서는 아키텍처 기반의 적응형 소프트웨어를 개발하기 위한 사전 작업으로 소프트웨어가 적응해야 하는 상황을 모델링하는 기법을 제안한다. 도메인 내의 공통점과 차이점을 분석하여 여러 어플리케이션을 개발하는 FORM[5]의 절차를 이용하여, 적응형 소프트웨어가 적응 해야 하는 여러 상황들을 휘쳐를 기반으로 하여 모델링 한다.

향후에는 문맥 분석, 휘쳐 분석, 참조 아키텍처 설계에 대한 보다 세부적인 절차와 각 모델간의 연관성이 보완되어야 한다. 또한, 참조 아키텍처를 기반으로 적응 상황 별로 실제 아키텍처를 설계, 구현하여 어플리케이션 수행 시에 아키텍처와 코드가 동적으로 적응하게 하는 방안에 대한 연구가 필요하다.

6. 참고문헌

[1] Robert Laddaga. Active Software. In Robert Laddaga Paul Robertson and Howard E. Shore, editors, Self-Adaptive Software. Spriger-Verlag, 2000.

[2] Robert Laddaga. Self-Adaptive Software sol baa 98-12. URL: http://www.darpa.mil/ipto/Solicitations/CBD_9812.html, 1988.

[3] Oriezy, P., Gorlick, M.M., Taylor, R.N., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., and Wolf, A. An Architecture-Based Approach to Self-Adaptive Software. In IEEE Intelligent Systems 14(3):54-62, May/June 1999.

[4] David Garlan. "Model-based Adaptation for Self-Healing Systems", ACM SIGSOFT Workshop on Self-Healing Systems (WOSS '02), November 18-19, 2002.

[5] C. Kang, S. Kim, J. J. Lee, K. Kim, E. Shin, M. Huh, FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, Annals of Software Engineering, 5:143-168, (1998).

[6] Kang, K. et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study (CMU/SEI-90-TR-021). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1990.