소프트웨어 진화를 위한 아키텍처 기반 프로그래밍

조병일, 윤현상, 이은석
성균관 대학교 전기전자컴퓨터공학과
e-mail : mes1an, wizehack, eslee@ece.skku.ac.kr

Architecture Based Programming for Software Evolution

Beoungil Cho, Hyun-sang Youn, Eunseok Lee

Department of Electrical and Computer Engineering,
Sungkyunkwan University

요 약

아키텍처를 기반으로 디자인 된 소프트웨어는 컴포넌트간의 낮은 결합력 때문에 재사용이나 부분적인 수정이 쉽다. 일반적으로 아키텍처는 디자인 단계에서 구성되며 아키텍처 디자인을 바탕으로 컴퍼넌트들을 구현한다. 그러나 프로그래밍 언어의 컴퍼넌트간 인터페이스는 아키텍처의 커넥터와 다르기 때문에 구현된 코드는 아키텍처 디자인을 있는 그대로 반영하지 못 한다. 결과적으로 차후 프로그램 코드의 수정이나 재사용이 아키텍처 디자인의 변경보다 복잡해진다. 본 논문에서는 아키텍처의 커넥터를 클래스를 통해 명확히 구현함으로써 아키텍처 디자인을 그대로 유지하는 코드 작성법을 제안한다.

1. 서론

소프트웨어 아키텍처는 소프트웨어를 컴포넌트, 커넥터 그리고 이들간의 제약사항을 이용하여 명세화한 디자인이다. 아키텍처로 명세화된 소프트웨어는 코드 레벨 이상의 재사용이 가능하고 컴포넌트간 결합력을 최소화 시켜 기존 시스템의 변경에도 효과적으로 대처할 수 있다. 아키텍처를 효과적으로 재사용하기 위해서는 아키텍처 모델과 구현 모델이 서로 일관성있게 관점을 유지하는 것이 중요하다. 따라서 아키텍처 진화와 관련된 연구에서는 구현코드와 아키텍처 모델을 동일하게 유지하는 기술이 중요하다. 그러나 코드화 된 프로그램은 디자인 단계에서 구성한 아키텍처를 유지하지 않는 경우가 많다. 특히 프로그램 코드는 커넥터의 속성을 제대로 나타내기 힘들기 때문에 구현모델과 디자인 모델의 아키텍처는 서로 다른 것이 일반적이다. 자가적응을 위해서는 실행 중인 프로그램이 디자인 단계에서 구성한 아키텍처를 유지하고 이에 대한 변경이 쉽도록 하는 것이 중요하다. 본 논문에서는 프로그램코드가 디자인 단계에서 정의한 아키텍처 레벨을 유지할 수 있도록 하는 프로그래밍 방법론을 제안하고자 한다.

2. 관련 연구

아키텍처를 이용하여 프로그램을 변경하기 위한 연구는 크게 두 가지 관점에서 볼 수 있다. 먼저 소프트웨어 아키텍처를 기술하기 위한 연구와 실제로 아키텍처를 변경하는 기술에 대한 연구이다. 아키텍처를 기술하기 위해 사용되는 방법으로는 XML 또는 정형명세언어를 이용하여 컴포넌트의 속성을 기술하는 방법이나, π-calculus 를 이용하여 시스템의 동작을 기술하는 방법이 있다.

아키텍처를 변경시키는 기술은 대표적으로 AOP 를 이용한 방법과 미들웨어를 이용한 방법이 있다. AOP 를 이용할 경우 동적으로 자가적응 하는 프로그램을 구현하기는 쉽다. 그러나 변경되는 프로그램의 아키텍처 성질을 적절한 Aspect 로 추출하지 못할 경우 프로그램이 실행되는 동안 아키텍처를 반영하기 어렵다. 미들웨어를 이용해 아키텍처를 변경시키는 방법은 시스템을 실행시간에 자가적응 시키기도 쉽고 실행코드가 아키텍처를 유지하도록 하는 것도 쉽다. 그러나 미들웨어 프레임 워크에서 정의된 인터페이스를 이용해야하기 때문에 커넥터의 구현이 제한적이다. 따라서 시스템 진화는 컴포넌트 변경을 통해서만 이루어져야 한다는 제약이 생긴다. 결과적으로 디자인 단계에서 정의한 아키텍처를 수정해야 한다.

3. 프로그래밍 방법론

JAVA 와 같은 객체지향 프로그래밍 언어의 클래스

이 논문은 2009 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. 2009-0077453)

는 아키텍처의 소규모 컴포넌트로 생각될 수 있으며 디자인 단계에서 정의한 아키텍처 모델의 컴포넌트 역시 클래스의 세트로 구현 될 수 있다. 그럼에도 불구하고 실행되는 프로그램이 아키텍처 성질은 나타내기 어려운 가장 큰 이유는 아키텍처 모델에서 자유자재로 변경 가능한 커넥터에 대응하는 구현 모델이 없기 때문이다. 대부분의 경우 커넥터는 클래스 내부의 메소드로 구현되어 컴포넌트와 커넥터 간의 독립적인 변경을 어렵게 한다. 이를 해결하기 위해 본 논문에서는 아키텍처의 커넥터를 컴포넌트와 독립된 클래스로 구현하여 디자인 단계의 아키텍처 모델을 그대로 유지할 수 있는 프로그래밍 방법을 제안한다.

앞서 말한 것과 같이 객체지향 언어의 클래스는 소프트웨어 아키텍처의 컴포넌트를 구현하기에 충분한 표현력을 가지고 있다. 객체지향언어는 클래스 사이의 메소드 호출을 통해 서로 통신한다. 자연스럽게 객체지향언어로 프로그래밍 된 아키텍처 모델의 커넥터는 메소드 호출로 표현되는 것이 일반적이다. 본 논문에서 제안 하는 방법론은 아키텍처 모델에서 정의 된 커넥터를 클래스로 정의 하여 컴퍼넌트와 커넥터를 독립적으로 구현하고자 한다. 표현력만을 생각했을 경우, 클래스는 커넥터의 행동을 기술하기에 충분하다. 그러나 컴퍼넌트가 커넥터와 연결하여 상호작용하기 위해서는 컴퍼넌트의 포트를 명확히 정의해야 한다. 컴포넌트의 포트를 명확히 함으로써 포트 사이를 연결하는 커넥터를 명확히 정의 할 수 있다. 클래스로 정의된 커넥터는 컴퍼넌트 클래스의 포트와 포트들을 동기화 하여 상호작용 할 수 있도록 돕는다. 위와 같은 커넥터와 컴퍼넌트의 정의를 통해 디자인 모델에서 정의한 아키텍처와 같은 관점의 프로그램 코드를 작성 할 수 있다.

본 논문에서 제안한 방법으로 아키텍처를 구현했을 경우 시스템의 규모가 커질수록 커넥터 구현에 소모되는 부하가 커진다. 모든 커넥터에 대하여 명시적으로 클래스를 구현하므로 구현 코드의 양이 늘어난다. 그러나 디자인과 구현의 관점을 일치시킴으로써 재사용성과 유지보수성을 높일 수 있으며, 향후 자가 적응과 같은 오토노믹컴퓨팅 기술의 기반기술로 사용될 수 있을 것으로 기대한다.

4. 결론

본 논문은 디자인 단계의 아키텍처 모델을 실제 구현에서도 적용할 수 있는 프로그래밍 방법론을 제안하였다. 이 방법론은 통하여 구현된 프로그램은 향후 변경 사항이나 추가사항이 생길 때 아키텍처 디자인이 수정 됨에 따라 아키텍처 모델을 그대로 구현에 적용할 수 있게 된다.

현재 아키텍처를 이용하여 소프트웨어를 진화시키는 연구가 활발히 진행되고 있다. 뿐만 아니라 아키텍처 모델을 이용하여 실행시간에 프로그램을 변경시키기는 연구 또한 활발하다. 우리는 현재 연구를 확장하여 개발자가 직접 아키텍처 변경을 코드로 구현하지 않아도 디자인 명세로부터 스켈레톤 코드를 자동 하는 단계까지 확장하고자 한다. 또한 최종적으로 이러한 아키텍처 진화에 따른 프로그램 변경이 실행시간에 이루어지는 자가적응 시스템을 완성하고자 한다.

참고문헌

[1] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund and Eli Gjorven, "Using Architecture Models for Runtime adaptability", ICT, Vol.23, Issue 2, page 62- 70, Mar. 2006

[2] Haimei Zhang, Kerong Ben and Zhixiang Zhang, "A Reflective Architecture-aware Framework to Support Software Evolution", ICYCS 2008. The 9th International Conference, page 1145-1149, Nov. 2008

[3] Ron Morrison, Dharini Balasubramaniam, Flavio Oquendo and Brian Warboys, "An Active Architecture Approach to Dynamic Systems Co-evolution" LNCS vol. 4758, page 2-10, Sep. 2007

[4] Paolo Falcarin and Gustavo Alonso, "Software Architecture Evolution through Dynamic AOP", LNCS vol.3047, page 57-73, May. 2004

[5] Stephen T. Albin, "The Art of Software Architecture : Design Methods and Techniques", Wiley, 2003

[6] James Coplien, Daniel Hoffman, and David Weiss, "Commonality and Variability in Software Engineering", Software, IEEE, Vol.15, Issue 6, page 37-45, Dec, 1998

[7] Jan Bosch, "Software Variability Management", Proceedings of the 26th International Conference on Software Engineering, page 720-721, 2004