UML 기반의 소프트웨어 아키텍처 표현방법

궁상환
백석대학교 정보통신학부
e-mail:kung@bu.ac.kr

Software Architecture Documentation based on UML

Sang Hwan Kung
BaekSeok University

요 약

본 논문에서는 UML을 이용하여 소프트웨어 아키텍처를 표현하는 방법을 소개하고, 복수의 뷰를 통한 아키텍처를 문서화하는 방법을 설명한다. 무엇보다 논문의 핵심은 UML의 다이어그램이 복잡하여 특별한 편집도구 없이는 표현이 불가능한 점을 반영하여, UML의 표현방법을 개선하고 이를 아키텍처 뷰의 표현에 적용한 점이라고 할 수 있다.

Key words : 소프트웨어 아키텍처, 아키텍처 문서화, UML

1. 개요

과거, 소프트웨어 아키텍처를 표현하는 방법은 설계자에 따라 매우 다양하게 이루어 졌다. 그러나 최근에 UML(Unified Modeling Language)이 등장함으로써, 소프트웨어 아키텍처의 표현을 위해 UML이 많이 활용되고 있다. 그렇지만, UML은 다이어그램이 복잡할 뿐만 아니라 표현방법이 지속적으로 변경되어 설계자에게 부담을 주는 경우가 많다. 특히, 다이어그램의 복잡성은 CASE 도구의 사용을 전제로 하며, 설계자들은 이 도구를 구매하여 활용방법을 익혀야 하는 부담을 부과한다.

논문에서는 UML을 이용하여 소프트웨어 아키텍처를 설계하는 방법과 UML을 표현방법을 단순화한 개선된 표현방법을 이용하여 아키텍처를 설계하는 방법을 설명하고자 한다.

2. 아키텍처 설계방법

과거의 전통적인 소프트웨어개발 방법론은 프로세스 중심적인 소프트웨어 개발방법에 대한 잘 설명해 주고 있다. 그러나 소프트웨어의 규모가 복잡해짐에 따라 이러한 방법론들은 아키텍처의 설계에 대해 추가적인 고려를 필요로 하게 되었다. 소프트웨어 아키텍처는 소프트웨어 시스템의 구조나 구조들의 집합이다. 이 구조는 소프트웨어의 구성요소와 각 요소의 가시적 특성, 그리고 요소들 간의 관계를 설명한다[4]. 아키텍처 중심의 개발방법은 소프트웨어 개발에서 아키텍처를 먼저 정립한 후, 아키텍처 베이스라인을 토대로 하여 구현하는 것을 말한다[1]. 아키텍처 설계의 핵심이라면, 기능적 요구사항이 되는 설계요소를 분해하고 설계할 때 요구품질에 적합한 패턴을 찾아 이 패턴에 분해결과를 매핑해 나간다는 것이다[2,5,8].

3. UML 기반의 아키텍처 표현

3.1 아키텍처 문서화

아키텍처는 표현되어야 하고 문서화되어야 한다. 그러나 오랫동안 아키텍처의 설계를 위한 표준화된 도구 없이, 원이나 선, 화살표 등을 이용한 비정형적인 방법이 사용되어 왔다. 비정형적인 방법은 아키텍처

의 표현에 대해 이해관계자들에게 통일된 이해를 주지 못하고, 다양한 관점에서 아키텍처를 표현하지 못하여 명확히 아키텍처를 제시하는 것은 불가능하였다. 전자의 문제에 대해서는 아키텍처의 표현을 위해 UML이 활용됨으로써 점차 해소되었고[3,6], 후자의 문제는 4+1 뷰와 같이 아키텍처를 복수의 관점에서 표현하는 방법에 의해 해결되게 되었다[7].

3.2 UML 기반의 아키텍처 뷰 표현

아키텍처의 문서화를 위한 핵심적인 뷰에는 논리적 뷰와 구현 뷰, 동시성 뷰, 배치 뷰가 포함된다. 이러한 복수의 뷰를 표현하기 위해, UML은 다양한 다이어그램을 활용한다. 예를 들어, 논리적인 설계요소를 표현하기 위해서는 패키지 다이어그램을 이용하며, 실행모듈이나 테이블과 같은 컴포넌트를 표현하기 위해서는 컴포넌트 다이어그램을 이용하고, 객체나 쓰레드를 표현하기 위해서는 협력 다이어그램을, 그리고 컴포넌트가 컴퓨팅 노드에 할당되는 것을 표현하기 위해서는 배치 다이어그램을 이용한다.

4. UML을 응용한 아키텍처 표현방법의 제안

4.1 기본 표현도구

UML 기반의 패키지 다이어그램이나 컴포넌트 다이어그램, 클래스 다이어그램, 그리고 배치 다이어그램은 서로 다른 모양을 하고 있고, 또 복잡해서, UML 다이어그램을 지원하는 자동화(CASE) 도구의 도움없이는 문서화가 용이하지가 않다.

이러한 불편에 대해, 본 연구에서는 UML의 표현방법을 보다 더 간편하게 개선하여 아키텍처의 문서화에 활용하고자 한다. 특히, 이 제안은 다음의 두 가지 개념을 기초로 출발한다.

첫째, UML에서 정의하는 다양한 설계요소를 하나의 박스로 표현한다. UML에서 박스는 클래스를 의미하지만 경우에 따라서는 개념적인 설계요소를 표현하기도 한다.

둘째, 설계요소의 유형을 표현하기 위해서는 스테레오타입을 이용한다. UML에서 스테레오 타입은 설계요소를 상세하게 표현하거나 부족한 설명을 추가할 때 사용한다.

특히, 소프트웨어 아키텍처가 하나의 뷰로만 표현되지 않기 때문에, 본 제안에서는 이러한 기본적인 두 개의 도구를 기초로 하여 아키텍처의 문서화에 필요한 다양한 뷰를 모두 표현할 수 있도록 하고자 한다.

먼저, 본 제안에서 사용하고자 하는 기본적인 표현도구를 소개하면 다음의 그림 1과 같다. 한편, 여기서 설계요소는 설계하고자 하는 뷰에 따라 논리적인 단위가 될 수도 있으며, 실행단위 또는 구현단위가 될 수도 있다.

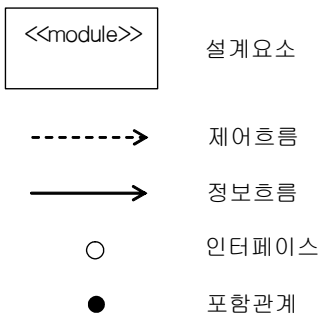

그림 1. 기본 표현 도구

4.2 아키텍처 뷰의 표현

본 제안에서는 중요한 아키텍처 뷰의 유형으로 데이터 뷰와 논리 뷰, 실행 뷰, 그리고 구현 뷰를 제안하고자 한다.

(1) 데이터 뷰의 표현

우리가 간과하기 쉬운 뷰는 데이터 뷰이다. 왜냐하면 데이터가 마치 프로그램과는 별도의 부분으로 간주되기 쉽기 때문이다. 그러나 데이터 뷰는 프로그램 특히, 컴포넌트의 기초가 되는 정보를 표현하는 중요한 뷰이다. 우리는 이 뷰를 위해 클래스의 표현을 이용하며, <<entity>>라는 스테레오 타입을 사용한다.

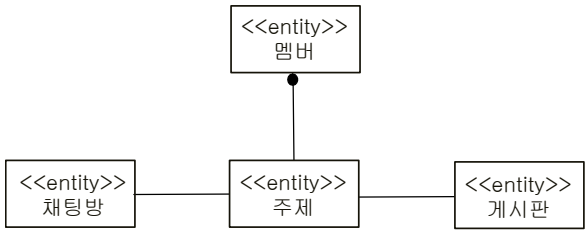

그림 2. 데이터 뷰의 표현

(2) 논리 뷰의 표현

통상 논리적 설계요소는 UML의 패키지 다이어그램을 사용하지만, 여기서는 단순한 박스와 스테레오타입을 이용하여 다음과 같이 정의하고자 한다. 따라서 설계요소는 그 설계요소의 타입을 설명하는 스테레오 타입의 명시(예: <<subsystem>>)와 그 설계

요소의 이름을 스테레오 타입 아래에 기록한다.

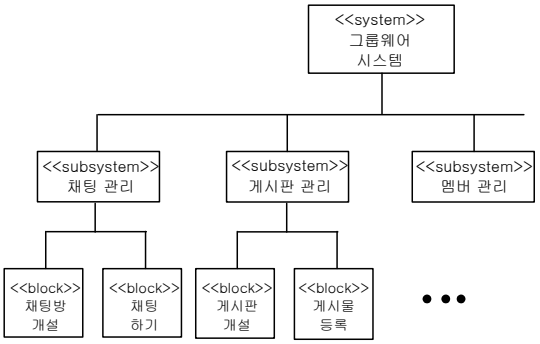

그림 3. 논리 뷰에서의 포함관계 표현

(3) 구현 뷰의 표현

구현 뷰는 논리적 설계요소가 프로그램으로 구현된 이후의 상태를 표현하는 뷰이다. 이 뷰의 표현은 설계요소를 위한 박스 안에 <<executables>>과 같은 설계요소의 유형을 스테레오 타입으로 기록하하여 컴포넌트임을 표시하도록 한다.

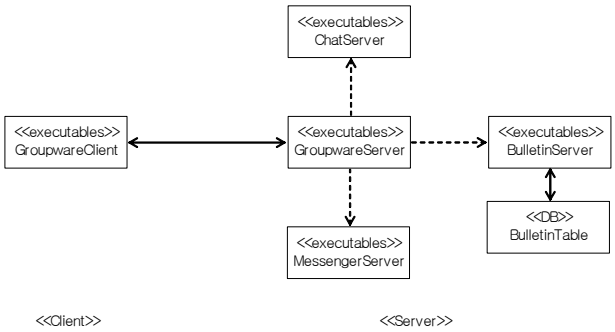

그림 4. 구현 뷰의 표현

3.3 실행 뷰의 표현

이 뷰는 UML의 표현과 거의 유사하다. 다만, 구현 뷰에서 배치의 관점을 표현하는 것과 마찬가지로 관련된 프로세스나 쓰레드의 주변에서 <<Client>>나 <<Server>>와 같은 명시를 할 수가 있다.

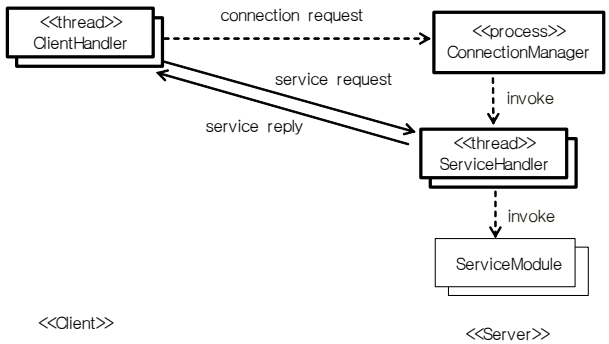

그림 5. 실행 뷰의 표현

4. 결론

논문에서는 아키텍처의 문서화에 필요한 뷰에 대해 살펴 보았다. 특히, 아키텍처의 문서화에 필요한 복수의 뷰를 표현하기 위해 UML이 활용되는 방법을 설명하였다. 그리고 UML의 복잡한 표현을 단순화하여 일반적인 편집기를 활용하여 쉽게 표현이 가능한 방법을 제안하고, 이 방법에 의해 아키텍처 뷰를 표현하는 방법을 보여 주었다.

제안된 방법에 의한 설계는 단순한 한편, UML로의 변환이 가능하다는 장점이 있다.

참고문헌

[1] Dean Leffingwell, Don Widrig, Managing Software Requirements - Unified Approach, Addison-Wesley, 2001.

[2] Felix Bachmann, Len Bass, Gay Chastek, Patric Donohoe, Fabio Perzzi, Architecture Based Design Method, Technical Report CMU/SEI-2000-TR-001, CMU Software Engineering Institute, 2000.

[3] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language, Addison Wesley, 2005. 5.

[4] Len Bass, Paul Clements, and Rick Kazman, Software Architecture in Practice, 1998.

[5] Less Bass and Rick Kazman, Architecture-Based Development, CMU Software Engineering Institute, Technical Report CMU/SEI-99-TR-007, ESC-TR-99-007, 1999.

[6] Paul Clements, etc, Documenting Software Architectures: Views and Beyond, Addison Wesley, 2002.

[7] Philippe B. Kruchten, "4+1 View model of Architecture", IEEE Software 12(6), pp.42-50, Rational Software(IBM), Nov. 1995.

[8] Rob Wojcik and et al, Attribute-Driven Design(ADD), Version 2.0, Technical Report CMU/SEI-2006-TR-023, CMU SoftwareEngineering Institute, 2006.