아키텍처 기반의 컴포넌트 조립을 지원하는 아키텍처 기술 언어의 설계와 구현

노성환[0] 신동익 전태웅
고려대학교 전산학과
(shroh[0], eastwing, jeon)@selab.korea.ac.kr

Design and implementation of an architecture description language that supports architecture-based component assembly

Sung-hwan Roh[0], Dong-ik Shin, Taewoong Jeon
Dept. of Computer Science, Korea University

요 약

컴포넌트 시스템은 잘 정의된 아키텍처를 기반으로 개발되어야 한다. 소프트웨어 아키텍처를 정확하고 엄밀하게 설계, 분석하기 위해서는 아키텍처 기술 언어(ADL)의 사용이 필요하다. 컴포넌트 시스템의 아키텍처 모델링에 ADL을 효과적으로 사용하기 위해서는 ADL로 기술된 명세 수준의 아키텍처 모델로부터 목표한 플랫폼에 부합하는 컴포넌트 시스템을 효율적으로 구현할 수 있어야 한다. 본 논문에서는 C2 스타일의 아키텍처에 기반한 컴포넌트 합성을 지원하는 ADL을 설계, 구현한 결과와 이를 EJB 컴포넌트들의 조립을 지원하는 도구의 아키텍처 기술 언어로 사용한 사례를 설명한다.

1. 서 론

컴포넌트 합성에 의한 응용 소프트웨어 시스템의 개발은 합성되는 컴포넌트들이 서로 정확하게 결합하여 작동할 수 있는 아키텍처를 기반으로 이루어져야 한다. 잘 정의된 아키텍처는 소프트웨어의 분해, 생성, 합성, 개조를 컴포넌트 단위로 가능하게 한다. 이는 특히, 복잡 방대하고, 다양한 플랫폼과 장기간 운영을 지원해야 하는 시스템들의 개발과 진화를 상당히 개선할 수 있다.

컴포넌트 시스템의 아키텍처를 정확하게 모델링하기 위해서는 아키텍처를 엄밀하게 표현할 수 있는 아키텍처 기술 언어(ADL: Architecture Description Language)의 사용이 필요하다. 현재 다양한 ADL들이 소개되어 있다[1, 2]. 소프트웨어 아키텍처를 분석, 설계하는데 ADL을 시험적으로 사용한 사례들도 많다. 하지만, 산업체에서 컴포넌트 기반의 상용 소프트웨어의 개발에 ADL이 본격적으로 사용된 사례는 흔치 않다. 이는 기존의 ADL로 기술된 아키텍처 모델과 실제로 구현된 시스템의 물리적인 아키텍처 사이에 존재하는 의미적인 격차를 좁히기 어렵기 때문이다.

ADL의 효과적인 사용을 위해서는 ADL로 기술된 명세 수준의 아키텍처 모델을 효율적으로 분석, 정제하여 목표한 컴포넌트 플랫폼에 부합하는 구현 시스템으로 완성할 수 있어야 한다. 본 논문에서는 C2[3] 스타일의 아키텍처를 기반으로 한 컴포넌트들의 합성을 지원하는 ADL을 설계, 구현한 결과와 이를 EJB[4] 컴포넌트 조립도구의 아키텍처 기술 언어로 사용한 사례를 설명한다.

2. 관련 연구

소프트웨어 아키텍처를 블록 다이어그램과 같은 종래의 아키텍처 표현 방식들보다 명시적이고 엄밀하게 기술할 수 있는 ADL들이 다수 소개되어 있다[1, 2, 3, 5, 6]. 현존의 ADL들은 각각 나름대로 시스템의 구조와 행위의 특정한 측면들에 대한 아키텍처 모델링에 유용한 표현 수단들을 제공한다. 하지만 아키텍처 수준에서 중요한 모든 측면들을 종합적으로 엄밀하게 기술할 수 있는 ADL은 아직 소개되어 있지 않다. 그리고 기존의 ADL로 기술된 명세 수준의 논리적인 아키텍처 모델을 EJB나 COM과 같은 컴포넌트 모델을 기반으로 한 물리적인 컴포넌트 시스템으로 정제, 구현하는 것이 쉽지 않다.

한편, 객체지향 소프트웨어의 분석, 설계에 보편적으로 사용되고 있는 UML[7]은 아키텍처 수준의 소프트웨어 모델링에도 유용한 표현 수단들을 풍부하게 제공한다. 하지만 UML은 논리적인 아키텍처 수준에서의 컴포넌트, 커넥터 및 이들의 상호 관계들을 명시적이고 직접적으로 표현할 수 있는 표기 형식들의 지원이 미흡하다. 현재 ADL로 기술되는 아키텍처 모델과 UML 모델 사이의 매핑 방법 등이 연구되고 있으나, ADL 모델과 UML 모델 사이의 semantic gap은 여전히 남아 있다[8, 9].

3. ADL의 설계와 구현

본 연구에서는 컴포넌트들의 합성 구조를 C2 스타일의 아키텍처로 기술할 수 있는 ADL을 설계, 구현하였다. ADL은 컴포넌트 명세를 컴포넌트들로 구성된 아키텍처 명세와 분리하여 기술할 수 있도록 컴포넌트 명세 언어(IDN: Component Interface Definition Notation)와 아키텍처 명세 언어(ADN: Architecture Description Notation)의 두가지 표기 형식으로 정의하였다.

① IDN – 포트 기반 인터페이스, 내부 메소드 및 메시지 전이 행위들로 정의되는 컴포넌트 명세 언어
② ADN – IDN으로 정의된 컴포넌트들과 C2 커넥터들의 연결 관계들로 정의되는 아키텍처 명세 언어

IDN 구문은 [그림 1]과 같이 top, bottom 포트로 구성된 컴포넌트 인터페이스의 메시지 선언 부분인 port construct, 컴포넌트 내부에서 호출되는 메소드 선언 부분인 methods construct, 그리고 메시지 송수신과 메소드 호출 관계로 정의되는 컴포넌트의 행위를 나타내는 behavior construct로 구성된다. Behavior construct에 속한

startup, cleanup, 그리고 received construct는 각각 해당 컴포넌트의 시작, 종료, 그리고 특정 메시지 수신 시 발생하는 메소드 호출과 메시지 전송들로 정의된 컴포넌트 행위들을 나타낸다.

```
<component> ::=
    <package>
    <import>
    component <comp_name> {
        ε |
        port top {
            out { <msg_decl_list> }
            in { <msg_ decl_ list> }
        }
        port bottom {
            out { <msg_ decl_ list> }
            in { <msg_ decl_ list> }
        }
        methods { <method_decl_list> }
        behavior {
            { startup { <invoked_methods> <generated_msgs> } ]
            [ cleanup {<invoked_methods> <generated_msgs> } ]
            [ received <received_msg> { <invoked_methods> <generated_msgs> } ]*
        }
        notes {
            [ <note_name> = <note_content> ; ]*
        }
    }
```

[그림 1] IDN Syntax

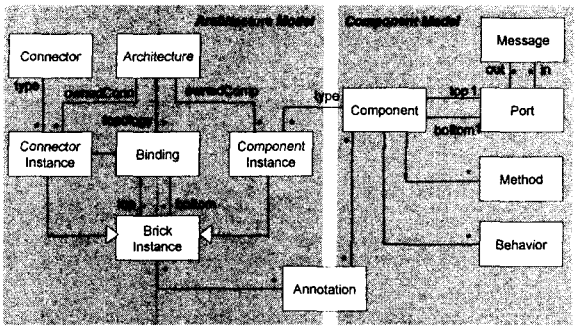
ADN 구문은 [그림 2]와 같이 아키텍처를 형성하는 컴포넌트와 커넥터의 인스턴스 선언 부분인 components, connectors construct들과 이들의 연결관계(binding)들로 정의되는 아키텍처 형세를 나타내는 topology construct로 구성된다. ADN의 components construct에서 선언되는 컴포넌트 인스턴스들의 타입은 IDN 구문으로 표현된 컴포넌트 명세에서 정의된다. 이와 반면, ADN에서 선언된 커넥터 인스턴스들의 타입은 C2 스타일에 따라 컴포넌트들 사이의 메시지 전송 통로의 역할을 갖는 C2 커넥터로 고정되어 있다.

```
<architecture> ::=
    <package>
    <import>
    architecture <arch_name> {
        ε |
        components {
            [ <context> <comp_name> <comp_inst_name_list> ; ]*
        }
        connectors {
            [ <conn_name> <conn_inst_name_list> ; ]*
        }
        topology {
            binding <conn_inst_name> {
                top = { <brick_inst_name_list> } ;
                bottom = { <brick_inst_name_list> } ;
            }*
        }
        notes {
            [ <note_name> = <note_content> ; ]*
        }
    }
```

[그림 2] ADN Syntax

위에서 정의한 ADL로 기술된 아키텍처 모델의 의미적인 구조는 [그림 3]과 같다. 이러한 ADL 모델의 기본적인 처리를 지원하는 ADL 처리기를 Java로 구현하였다[10]. 구현된 ADL 처리기는 정의된 ADL로 기술된 아키텍처 모델의 편집, 파싱, 스타일 검사, 및 언파싱(unparsing)을 수행하는 아래의 도구들을 포함한다.

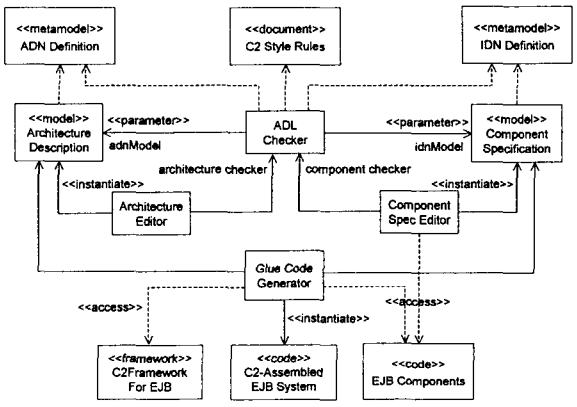
① ADL 편집기 – 정의된 ADL로 기술된 아키텍처 모델의 편집, 저장, 출력을 지원한다.
② ADL 파서 – ADL로 기술된 아키텍처 모델의 구문 오류를 검사하고 파스 트리를 생성한다.
③ ADL 스타일 검사기 – ADL 모델을 분석하여 C2 스타일을 위반하는 형세 오류와 타입 오류를 검사한다.
④ ADL 언파서 – 파싱된 내부 추상 형태로 표현된 아키텍처 모델을 ADL 구문으로 변환한다.



[그림 3] ADL 모델의 의미 구조

4. ADL을 사용한 컴포넌트 조립 도구 지원

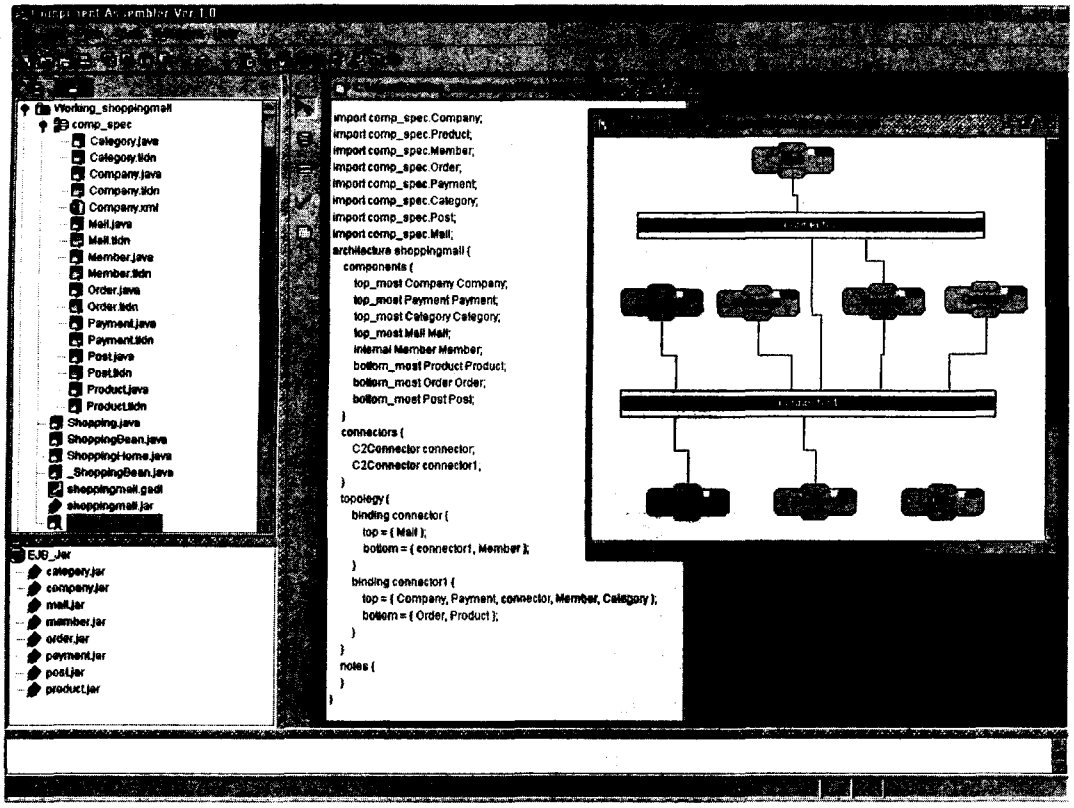
본 연구에서 설계, 구현한 ADL은 EJB 컴포넌트들을 C2 스타일의 아키텍처로 조립하는 도구인 COBALT (COmponent-Based Application deveLopment Toolset) Assembler [11]의 아키텍처 기술 언어로 사용되었다. 구현된 ADL 처리기는 COBALT 어셈블러에서 ADL 검사기로 사용되었다. [그림 4]는 본 ADL이 사용된 COBALT 어셈블러의 개략적인 전체 구조를 보여준다.



[그림 4] COBALT 어셈블러의 전체 구조

[그림 4]에서와 같이 COBALT 어셈블러는 ADL 검사기 외에도 다이어그램 형태의 아키텍처 편집을 지원하는 아키텍처 편집기, EJB 컴포넌트 정보를 참조하여 이에 대응하는 C2 컴포넌트 명세의 편집을 지원하는 컴포넌트 명세 편집기, 그리고 EJB 컴포넌트들을 C2 아키텍처로 조립하는데 필요한 접속 코드를 생성하는 접속코드 생성기를 포함한다.

ADL 검사기는 아키텍처 에디터로 편집된 아키텍처 모델과 컴포넌트 명세 편집기로 작성된 컴포넌트 모델의 구문과 의미 상의 오류 여부를 검사한다. ADL 검사기에 의해 모델 검사가 완료된 아키텍처 모델과 컴포넌트 모델들은 접속코드 생성기가 EJB 컴포넌트들을 C2 컴포넌트로 변환하기 위한 wrapper 코드를 생성하고 이들을 C2 아키텍처로 조립하여 합성 EJB를 만드는 데 사용된다.



[그림 5] COBALT 어셈블러에서 ADL을 사용한 아키텍처 모델링의 예

[그림 5]는 COBALT 어셈블러에서 쇼핑몰 시스템의 아키텍처를 본 논문에서 정의한 ADL로 모델링하고 있는 모습을 보여준다. [그림 5]에서 다이어그램 형태의 아키텍처 명세는 아키텍처 편집기에 의해, 텍스트 형태의 아키텍처 명세는 ADL 처리기에 내장된 ADL 텍스트 편집기에 의해 각각 편집되며, 이들 사이의 동기화를 위한 상호 변환이 자동으로 이루어진다. 위와 같이 시험 사용한 결과, 본 연구에서 설계, 구현한 ADL은 아키텍처 기반의 EJB 컴포넌트 조립을 위한 도구 지원에 필요한 아키텍처 모델의 표현과 처리를 잘 지원함을 확인하였다.

5. 결론 및 향후 연구

본 논문에서는 C2 아키텍처 기반의 컴포넌트 조립을 지원하는 ADL 을 설계, 구현한 결과와 이를 EJB 컴포넌트들의 조립을 지원하는 도구의 아키텍처 기술 언어로 사용한 사례를 소개하였다. 향후, 본 논문에서 정의한 ADL을 C2 스타일이 아닌 아키텍처들의 기술도 지원할 수 있도록 확장할 계획이다.

참고 문헌

[1] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996

[2] N. Medvidovic and R.N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Trans. on Software Eng., 26(1), Jan. 2000, pp. 70-93

[3] R.N. Taylor, et al., "A Component- and Message-Based Architectural Style for GUI Software", IEEE Trans. Software Eng., 22(6), June 1996, pp. 390-406

[4] Sun Microsystems, Enterprise JavaBeans Specification

[5] D.C. Luckham, et al., "Specification and Analysis of System Architecture Using Rapide", IEEE Trans. Software Eng., 21(4), Apr. 1995

[6] J. Magee and J. Kramer, Dynamic Structure in Software Architectures, Proceedings of the 4th ACM SIGSOFT Symp. Foundations of Software Eng., Oct. 1996, San Francisco, CA, USA pp. 3-14

[7] OMG Unified Modeling Language Specification, version 1.4, 2001

[8] D. Garlan, et al., "Reconciling the Needs of Architectural Description with Object-Modeling Notations", Proc. 3rd Int'l Conf. on the Unified Modeling Language, Oct. 2000, York, UK

[9] N. Medvidovic, et al., "Modeling Software Architectures in the Unified Modeling Language", ACM Trans. Software Engineering and Methodology (TOSEM), 11(1), Jan. 2002, pp. 2-57

[10]신동익, 노성환, 최재각, 전태웅, "ADL 처리기의 설계와 구현", 한국정보과학회 학술발표회 논문집, Oct. 19-20, 2001, pp. 382-384

[11]이승연, 이지현, 권오천, 신규상, "아키텍처 스타일 기반의 컴포넌트 조립 및 지원도구의 개발", 한국정보과학회 학술발표회 논문집, Oct. 19-20, 2001, pp. 370-372