C2 스타일의 아키텍처 기술을 지원하는 ADL 정의

노성환[0], 신동익, 최재각, 전태웅
고려대학교 전산학과
{shroh, eastwing, choi, jeon}@selab.korea.ac.kr

Defining an ADL that supports the description of C2 style architectures

Sung-hwan Roh[0], Dong-ik Shin, Jae-kak Choi, Taewoong Jeon
Dept. of Computer Science, Korea Univ.

요 약

소프트웨어를 아키텍처 수준에서 정확하고 엄밀하게 설계, 분석하는 것이 점차 중요해짐에 따라 사용이 쉬우면서 기술적으로 성숙된 아키텍처 기술 언어(ADL Architecture Description Language)의 필요성이 커지고 있다. 본 논문에서는 컴포넌트 기반의 도메인 아키텍처 모델링 시 C2 스타일의 아키텍처 기술을 지원하는 ADL을 정의한다. 본 논문에서 정의한 ADL은 자바(Java)와 유사한 구문으로 컴포넌트 명세와 아키텍처 명세를 분리하여 기술할 수 있는 표기 형식을 제공한다.

1. 서 론

다수의 응용 개발에 재사용 가능한 컴포넌트를 기반으로 한 소프트웨어 개발 방식이 최근 빠르게 확산되고 있다. 컴포넌트의 설계와 재사용은 아키텍처 수준에서 이루어져야 컴포넌트 시스템의 효율적인 개발과 개발된 시스템의 품질의 향상을 가져올 수 있다. 따라서 소프트웨어를 아키텍처 수준에서 정확하고 엄밀하게 설계, 분석하는 것이 점차 중요해지고 있다.

소프트웨어 아키텍처는 아키텍처 기술 언어(ADL:Architecture Description Language)를 사용하여 기술되어야 정확하고 엄밀한 아키텍처 모델링과 아키텍처에 기반한 시스템의 분석, 정제, 검증이 가능하다. 아키텍처 기술 언어(ADL)와 ADL 지원 환경에 대한 연구는 90년대 초반부터 외국에서 활발히 진행되어 왔다[1,2]. 그 결과, 현재 다양한 ADL들과 ADL 지원 도구들이 소개되어 있다[3, 4, 5, 6, 7].

현존의 ADL들은 대부분 외국의 대학이나 연구기관에서 연구 프로젝트의 결과로 얻어진 것들로서, 프로토타입 수준의 ADL 지원도구들도 함께 제작되어 있거나 개발이 진행 중에 있다. 아키텍처 모델링에 ADL을 시험적으로 사용한 사례들은 많다. 그러나 산업체에서 실제 상용 소프트웨어의 개발에 ADL이 사용된 예는 흔치 않다. 소프트웨어 개발에서 아키텍처의 중요성이 높아지고 있음에 비추어 사용이 용이하면서 기술적으로 성숙된 ADL 및 이의 지원도구의 실용화가 시급한 실정이다. 하지만 국내에서는 ADL을 개발하거나 ADL을 사용한 소프트웨어 개발 사례가 아직 없다.

본 논문에서는 컴포넌트 기반의 도메인 아키텍처 모델링 시 C2 스타일의 아키텍처 기술을 지원하는 ADL을 정의한다. 본 논문에서 정의한 ADL은 자바(Java)와 유사한 구문으로 컴포넌트 명세와 아키텍처 명세를 분리하여 기술할 수 있는 표기 형식을 제공한다. 분리되어 기술된 아키텍처 모델 요소들은 자바에서와 같은 계층적인 패키지 단위로 구성된다. 다른 패키지에 속한 필요한 타입들은 import하여 사용한다.

2. C2 아키텍처 스타일

C2 스타일[8]은 UCI(University of California in Irvine)에서 본래 GUI를 갖는 응용 소프트웨어 개발을 지원하기 위하여 설계한 아키텍처 스타일이지만 다른 타입의 응용 소프트웨어 개발도 잘 지원한다. C2 스타일의 컴포넌트는 두 개(top, bottom)의 포트를 통해 들어오는 메시지(incoming message)에 대해 내부적으로 선언된 메소드(들)을 호출(invocation)하거나 top 또는 bottom 포트를 통해 나갈 메시지(outgoing message)를 생성한다. 컴포넌트가 처리 또는 생성하는 메시지는 top 포트를 통해 나가거나 bottom 포트를 통해 들어오는 요청(request) 메시지와 top 포트를 통해 들어오거나 bottom 포트를 통해 나가는 공고(notification) 메시지로 구분된다. C2 스타일의 모든 컴포넌트들은 메시지 교환(message exchange)을 통해 상호작용을 한다. C2 스타일의 커넥터는 컴포넌트들 간에 교환되는 메시지를 라우팅(routing), 브로드캐스팅(broadcasting) 또는 메시지 필터링(message filtering)하는 역할을 한다.

C2 스타일의 컴포넌트는 top포트 또는 bottom 포트를 통해서만 커넥터의 bottom포트 또는 top 포트에 연결될 수 있다. 따라서 C2 스타일에서는 컴포넌트 간의 직접 통신은 허용되지 않는다. 그리고 커넥터는 복수 개의 컴포넌트들 또는 또 다른 커넥터들과 연결될 수 있다.

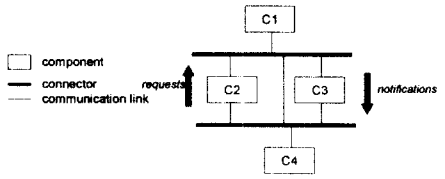

그림 1: C2 architectural style

3. C2 스타일 기반의 아키텍처 기술 언어

본 연구팀은 C2 스타일 기반의 ADL을 1) 컴포넌트 인터페이스 명세 언어(IDN:Interface Definition Notation)와 2) IDN으로 정의된 컴포넌트들과 C2 커넥터들 사이의 바인딩들로 구성된 아키텍처 형세(topology)를 기술하는 아키텍처 명세 언어(ADN:Architecture Definition Notation)로 나누어 설계하고 각 언어의 구문(syntax) 및 의미(semantics)를 정의하였다. ADL의 구문은 순환적 내림차순 파싱(recursive-descent parsing)이 가능하도록 LL(1) 문법으로 설계하였다.

본 장에서는 본 연구에서 정의한 컴포넌트 명세 언어와 아키텍처 명세 언어의 구문들에 대하여 기술하고, UCI의 C2SADL[9]에서 예제로 사용했던 StackADT 컴포넌트와 스택 시각화 시스템 아키텍처인 StackVisualization를 본 연구에서 정의한 ADL 구문으로 명세한다.

[0] 본 논문은 한국전자통신연구원(ETRI)의 위탁과제로 수행되었음

3.1 컴포넌트 명세 언어- IDN(Interface Definition Notation)

컴포넌트 명세 언어 즉, IDN은 컴포넌트의 포트 메시지와 메소드의 선언 부분, 그리고 선언된 메시지와 메소드를 통해 컴포넌트의 행위를 나타내는 부분으로 구성되어 있다.

(그림2)는 컴포넌트 명세 언어-IDN의 구문(syntax)이다. (그림2)에서 포트 부분(port construct)은 컴포넌트의 top 포트와 bottom 포트를 통해 나가거나 들어오는 메시지 리스트를 선언한다. 메소드(method) 부분은 top 또는 bottom 포트를 통해 들어오는 메시지에 대해 컴포넌트가 호출할 수 있는 메소드 리스트를 선언한다. 그리고 행위 부분(behavior construct)은 컴포넌트의 동작 시작, 종료, 또는 실행 시 나타나는 행위를 명시한다.

```
<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> } ]*
        }
    }
```

그림 2: IDN Syntax of ADL

컴포넌트의 동작 시작, 또는 종료 시에 행해지는 행위는 선택적(optional)이고 포트를 통해 들어오는 메시지에 의해 행해지는 행위가 아니다. 반면에, 실행 시에 행해지는 행위는 컴포넌트의 특정 포트에 들어오는 메시지에 의해 정의된 메소드(들)을 호출하거나 정의된 포트를 통해 메시지(들)을 내보낸다.

```
package arch.stack;
import arch.stack.datatype Element;

component StackADT {
    port top {
        out {}
        in {}
    }
    port bottom {
        out {
            elementPushed(Element value),
            elementPopped(Element value),
            topElement(Element value);
            stackEmpty();
        }
        in {
            pushElement(Element value);
            popElement();
            getTopElement();
        }
    }
    methods {
        void push(Element);
        Element pop();
        Element getTop();
        boolean isEmpty();
    }
behavior {
    received bottom.pushElement(Element) {
        invoke push(Element);
        generate bottom.elementPushed();
    }
    received bottom.popElement() {
        invoke isEmpty(), pop();
        generate bottom.stackEmpty()
                 | bottom.elementPopped();
    }
    received bottom.getTopElement() {
        invoke isEmpty(), getTop();
        generate bottom.stackEmpty()
                 | bttom.topElement(Element);
    }
}
```

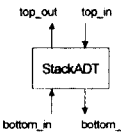

그림 3: Example of IDN Description

(그림3)는 StackADT[9] 컴포넌트 명세를 본 연구에서 정의한 IDN 구문으로 기술한 예이다.

StackADT는 스택(stack)에 대한 추상데이터 타입을 제공한다. (그림3)에서 StackADT 컴포넌트는 top 포트로 송수신되는 메시지는 없는 반면에 bottom 포트로 송수신되는 메시지들이 있다는 것을 알 수가 있다. bottom 포트로 송수신되는 메시지들은 port bottom 부분의 out과 in 부분에 선언되어 있다. methods 부분에는 스택 작업을 위해 내부적으로 호출될 수 있는 메소드들이 선언되어있다. 그리고 behavior 부분에는 특정 포트로 수신된 메시지에 의해 행해지는 행위, 즉 메소드를 호출하거나 top 또는 bottom 포트 외부로 나갈 송신 메시지를 생성하는 행위가 received 부분들에 명시되어 있다. (그림3)의 StackADT 컴포넌트 예에서는 컴포넌트 동작 시작 또는 종료 시 필요한 행위는 명시되어 있지 않다.

3.2 아키텍처 명세 언어- ADN(Architecture Definition Notation)

아키텍처 명세 언어 즉, ADN은 아키텍처의 컴포넌트와 커넥터의 인스턴스 선언 부분, 그리고 선언된 컴포넌트와 커넥터의 인스턴스들 간의 연결 관계를 나타내는 부분으로 구성되어 있다. ADN에서 선언되는 컴포넌트 인스턴스들의 타입은 IDN 구문에서 정의된다.

```
<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 {
            [ <brick_inst_name> = <note> ; ]*
        }
    }
```

그림 4: ADN Syntax of ADL

(그림4)는 아키텍처 명세 언어-ADN의 구문(syntax)이다. ADN 구문의 컴포넌트 선언 부분은 아키텍처 명세에서 사용될 컴포넌트 인스턴스(들)을 열거한다. 각 컴포넌트 인스턴스는 아키텍처 상의 위치(top_most, internal, 또는 bottom_most), 컴포넌트 타입, 그리고 컴포넌트 인스턴스명을 갖는다. 만약 컴포넌트 인스턴스의 아키텍처 상의 위치가 생략되었다면 해당 컴포넌트 인스턴스의 아키텍처 상의 위치는 internal로 간주된다. ADN 구문의 커넥터 선언 부분은 아키텍처 명세에서 사용될 커넥터 인스턴스(들)을 열거한다. 각 커넥터 인스턴스는 커넥터 타입, 커넥터 인스턴스명을 갖는다.

ADN의 형세(topology) 부분은 아키텍처 명세에서 선언된 컴포넌트와 커넥터의 인스턴스들 간의 연결 관계를 ADN의 바인딩(들)로 표현한다. ADN의 각 바인딩은 커넥터 인스턴스명과 해당 커넥터 인스턴스의 top과 bottom에 연결되어 있는 컴포넌트 인스턴스명 또는 또 다른 커넥터 인스턴스명들로 구성된다. ADN의 노트(note) 부분은 아키텍처 명세의 컴포넌트 또는 커넥터 인스턴스들에 대한 주석(annotation)들로 구성된다.

(그림 5)는 스택 시각화 시스템[9]을 본 연구에서 정의한 ADN 구문으로 기술한 예이다. (그림5b)는 블록 다이어그램으로 표현된 (그림5a)의 스택 시각화 응용을 위한 C2 스타일 아키텍처를, 본 연구에서 정의한 ADN으로 기술한 아키텍처 명세이다. stackADT는 (그림3)에서 설명한 StackADT 타입의 인스턴스이고 stackArtist1과 stackArtist2는 StactArtist 타입의 인스턴스이다. stackArtist1과

stackArtist2는 추상데이터 모델인 StackADT의 시각화 정보를 갖는 뷰 모델을 관리하는 컴포넌트 인스턴스들로서 StackADT 인스턴스에 들어 있는 정보를 다른 형태로 시각화 해준다. 그리고 graphicBinding은 GraphicsServer 컴포넌트의 인스턴스로서 상위 계층의 StackArtist 컴포넌트 타입의 인스턴스들이 사용할 그래픽 객체들을 제공하는 하위 수준의 그래픽 제공자이다.

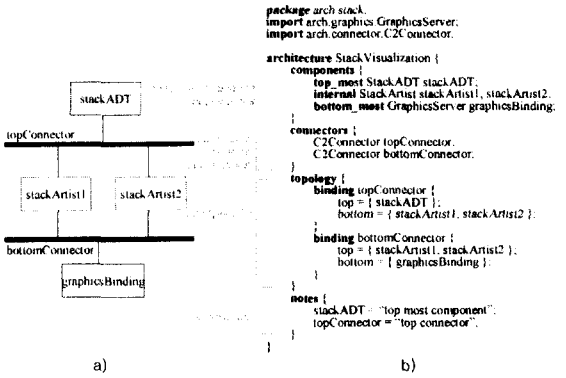

그림 5: Example of ADN Description

3.3 ADL 모델 명세들의 구성 방식

본 연구팀이 정의한 ADL은 자바와 동일한 의미의 package문과 import문을 제공한다. package문을 이용해 관련 있는 아키텍처 명세 또는 컴포넌트 명세들을 계층적으로 명명된 패키지 단위로 그루핑하고, import문을 이용해 다른 패키지에 포함되어 있는 필요한 타입들을 import한다.

예를 들면, 스택 시각화 시스템의 아키텍처 명세인 StackVisualization(그림5)은 StackADT(그림3)와 StackArtist의 컴포넌트 명세와 동일한 arch.stack 패키지에 속해 있다. 그리고 StackVisualization은 arch.graphics 패키지에 속한 GraphicsServer 컴포넌트 명세와 arch.connector 패키지에 속한 C2Connector 커넥터 명세를 import한다. StackADT 컴포넌트 명세는 arch.stack.datatype 패키지에 속한 객체 데이터 타입인 Element 명세를 import한다.

(그림6)는 이와 같은 (그림3)과 (그림5)의 아키텍처 명세에 나타난 컴포넌트, 커넥터, 그리고 데이터 타입들의 패키지 구성과 import 관계를 보여주는 다이어그램이다.

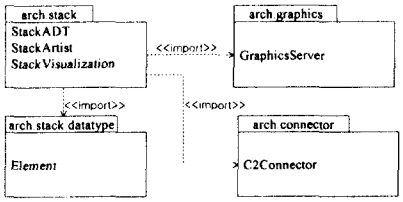

그림 6: Example ADL Description Packages

4. 기존 연구와의 비교

본 연구와 직접 관련 있는 UCI의 C2SADL[9]은 컴포넌트 인터페이스를 명세하기 위한 IDN(Interface Definition Notation), 아키텍처에 대한 선언적인(declarative) 명세를 위한 ADN(Architecture Definition Notation), 그리고 아키텍처의 동적인 변화를 표현하기 위한 ACN(Architecture Construction Notation)을 제공하고 있다.

본 연구에서 정의한 ADL은 UCI의 C2SADL이 제공하는 IDN 및 ADN과 의미적으로 동등한 수준의 C2 스타일 아키텍처의 기술을 지원하면서 자바에 친숙한 설계자가 쉽게 사용할 수 있는 구문으로 설계되어 있다. 현재, 본 연구에서 정의된 ADL에서는 C2SADL에서 아키텍처의 동적 변화를 표현하기 위해 제공하는 ACN은 지원하지 않는다. 반면에, 자바의 package문과 import문을 도입하여 아키텍처 모델 요소들을 패키지 단위로 그루핑하고, 다른 패키지에 있는 컴포넌트 명세나 데이터 타입을 import할 수 있게 하였다.

5. 결론 및 향후 연구

본 논문에서는 C2 스타일의 아키텍처를 자바와 유사한 구문으로 컴포넌트 명세와 아키텍처 명세를 분리하여 기술할 수 있도록 정의한 아키텍처 기술 언어에 대하여 설명하였다. 정의된 ADL은 ETRI에서 개발 중인 컴포넌트 조립 및 생성 지원 도구의 아키텍처 기술 언어로 사용될 예정이다

본 연구팀은 또한 본 논문에서 정의한 ADL을 지원하는 편집기와 파서를 개발하였다. 그리고 정의된 ADL로 기술된 아키텍처 모델의 타입 및 형세를 검사할 수 있는 모델 검사기를 개발 중에 있다.

향후에는, 현재 정의한 C2 스타일의 ADL를 아래에 열거한 표현 능력을 가질 수 있게 확장, 일반화하여 소프트웨어를 아키텍처 수준에서 설계, 분석, 정제, 진화하는데 효과적으로 사용할 수 있는 ADL로 발전시킬 계획이다.

- 아키텍처 명세에서 커넥터의 의미를 컴포넌트와 동등한 수준에서 직접적으로 표현할 수 있는 능력.
- 서브 아키텍처를 갖는 합성 컴포넌트(composite component)를 표현할 수 있는 능력.
- 아키텍처 수준에서 컴포넌트와 시스템 행위의 의미를 정확히 표현할 수 있는 능력.
- 아키텍처의 동적 변화를 표현할 수 있는 능력.
- C2가 아닌 스타일의 아키텍처도 기술할 수 있는 능력.

참고문헌

[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 Transactions on Software Engineering, Vol. 26, No. 1, January 2000, pp. 70-93

[3] J. Magee and J. Kramer, "Dynamic Structure in Software Architectures", Proceedings of the 4th ACM SIGSOFT Symposium on Foundations on Software Engineering, October 16-18, 1996, San Francisco, CA, pp. 3-14

[4] D. C. Luckham, L. M. Augustin, J. J. Kenney, J. Vera, D. Bryan, and W. Mann, "Specification and Analysis of System Architecture Using Rapide", IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995

[5] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for Software Architecture and tools to Support Them", IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995, pp. 314-335

[6] D. Garlan, R. Allen, and J. Ockerbloom, "Exploiting Style in Architectural Design Environments", Proceedings of SIGSOFT '94 Symposium on the Foundations of Software Engineering, December 1994

[7] D. Garlan, R. Monroe, and D. Wile, "Acme: An Architecture Description Interchange Language", Proceedings of CASCON'97, Nov. 1997, pp. 169-183

[8] R.N. Taylor, N. Medvidovic, K.M. Anderson, E.J. Whitehead Jr., J.E. Robbins, K.A. Nies, P. Oreizy, and D.L. Dubrow, "A Component- and Message-Based Architectural Style for GUI Software", IEEE Transactions on Software Engineering, Vol. 22, No. 6, June 1996, pp. 390-406

[9] The C2 Style, http://www.ics.uci.edu/pub/arch/c2.html, Information and Computer Science, University of California, Irvine.