ADL 모델을 Java 구현 모델로 매핑하는 방법

김수일[0] 전태웅
신한은행시스템, 고려대학교
soo2lee@shinhan.com[0], jeon@korea.ac.kr

A Method for mapping an ADL Model to a target Java implementation model

Soo-Il Kim[0] Tae-Woong Jeon
Shinhan Bank System, Korea University

요 약

아키텍처 기술 언어(ADL : Architecture Description Language)는 상위 추상화 수준에서의 소프트웨어 시스템의 구조와 행위를 기술하는데 필요한 아키텍처 모델 요소들에 대한 직접적인 표현 수단을 제공한다. ADL은 특히 명세 수준의 논리적인 아키텍처를 모델링, 분석하는데 유용하다. 그렇지만 ADL로 기술된 상위 수준의 아키텍처 명세가 하위 수준의 구현 시스템으로 어떻게 상세화 되는지 알기가 쉽지 않다. 즉, 상위 수준의 아키텍처 명세와 하위 수준의 아키텍처 구현 사이에 의미적 차이가 존재한다. 본 논문은 이러한 의미적 차이를 효율적으로 줄일 수 있는 방법을 찾기 위한 일차적인 연구 결과로서, C2 스타일 기반의 ADL로 기술된 아키텍처 모델을 Java 구현 코드로 자동 변환하는 방법을 제안한다.

1. 서 론

현재 다양하게 소개되어 있는 아키텍처 기술 언어(ADL: Architecture Description Language)들[1]은 상위 추상화 수준에서의 소프트웨어 시스템의 구조와 행위를 엄밀하게 설계, 분석하는데 유용하다. 그러나 ADL로 기술된 명세 수준의 아키텍처 모델이 구현 수준의 물리적인 아키텍처로 어떻게 상세화 되는지 알기가 쉽지 않다. 이는 상위 수준의 아키텍처 명세와 하위 수준의 아키텍처 구현 사이에 의미적 차이가 존재하기 때문이다.

본 논문은 이러한 의미적 차이를 줄일 수 있는 방법을 찾기 위한 일차적인 연구 결과로서 C2[2] 스타일 기반의 ADL 모델을 이를 구현하는 Java 프로그램 코드로 자동 변환하는 방법을 제시한다. 제안한 방법은 ADL 모델이 가지는 의미를 구현단계의 프로그램 코드로 매핑하는 규칙들을 기반으로 한다. 그리고 제안한 매핑 규칙들에 따라 ADL 모델의 구현 코드를 자동 생성하는 도구를 개발하였다.

2. 선행 연구

2.1 C2 아키텍처 스타일

UCI(University of California Irvine)에서 제안한 C2 스타일은 그림 1과 같은 메시지와 컴포넌트 기반의 다층 구조로 구성된 아키텍처 스타일이다. C2는 원래 GUI(Graphic User Interface) 요소를 갖는 소프트웨어를 지원하기 위해 만들어진 아키텍처 스타일이지만 다른 타입의 응용 소프트웨어들의 개발도 잘 지원한다. C2 스타일의 아키텍처에 핵심적인 구성 요소들은 다음과 같다.

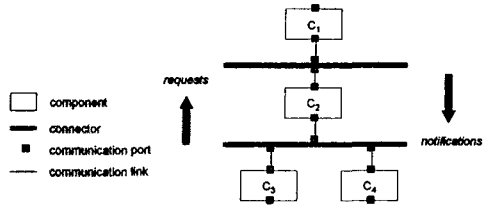

그림 1: C2 스타일의 아키텍처

- 컴포넌트 - top, bottom으로 구분된 2개의 포트를 인터페이스로 제공한다. 컴포넌트의 행위는 포트를 통해 수신되는 각 메시지에 대한 메시지 송신과 내부 메소드 호출로 정의된다.
- 커넥터 - top, bottom으로 구분된 불특정 개수의 포트들을 인터페이스로 제공한다. 컴포넌트들을 연결하고 연결된 컴포넌트들 사이의 메시지 전송 통로의 역할을 수행한다.
- 포트 - 컴포넌트와 커넥터의 연결 부위이다. 송수신 되는 메시지들이 정의되어 있다.
- 메시지 - request와 notification 두 가지의 메시지 타입을 갖는다.

2.2 C2 Class Framework

UCI는 그들이 정의한 C2 스타일의 아키텍처에 공통적인 부분들을 클래스 프레임워크로 구현하였다[3]. 이것은 컴포넌트, 커넥터, 메시지, 포트와 같은 C2 스타일의 핵심 개념들을 C++, Java 등의 프로그래밍 언어로 구현한 추상 클래

스들로 구성되어 있다. C2 스타일의 특정 아키텍처는 C2 클래스 프레임워크를 개조, 확장함으로써 구현할 수 있다. 그림 2는 Java로 구현된 C2 프레임워크의 UML 클래스 다이어그램이다.

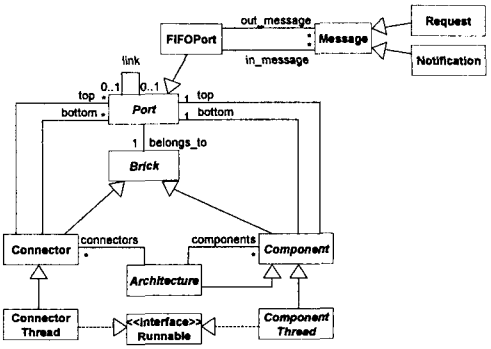

그림 2: C2 Java 클래스 프레임워크

2.3 C2 스타일 기반의 ADL

본 논문에서는 컴포넌트들의 합성 구조를 C2[2] 스타일의 아키텍처로 기술할 수 있게 정의한 ADL[4]을 사용하였다. 본 논문에서 사용한 ADL은 컴포넌트 명세의 표기 형식인 IDN(Interface Definition Notation)과 아키텍처 명세의 표기 형식인 ADN(Architecture Description Notation)으로 구성되어 있으며, UCI에서 정의한 C2SADL[5]의 IDN과 ADN을 Java와 유사한 구문으로 재정의한 것이다.

2.3.1 IDN(Interface Definition Notation)

컴포넌트 명세 언어인 IDN은 포트 기반 인터페이스, 내부 메소드의 선언 부분, 그리고 메시지 전이 행위 부분들로 구성되어 있다[그림3]. 포트 부분은 컴포넌트의 top 포트와 bottom 포트를 통해 나가거나 들어오는 메시지 타입들을 선언한다. 메소드 부분은 top 또는 bottom 포트를 통해 들어오는 메시지에 대해 컴포넌트가 호출할 수 있는 메소드들을 선언한다. 그리고 행위 부분은 컴포넌트의 시작, 종료, 및 메시지 수신 시 발생하는 메소드 호출과 메시지 송신으로 표현되는 컴포넌트 행위를 명시한다.

```
<component> ::=
    package <package_name>
    [ import <import_list> ]*
    component <comp_name> {
        ε |
        port top {
            out { [ <msg_name> ( <param_list> ); ]* }
            in { [ <msg_name> ( <param_list> ); ]* }
        }
        port bottom {
            out { [ <msg_name> ( <param_list> ); ]* }
            in { [ <msg_name> ( <param_list> ); ]* }
        }
        methods { [ <method_name> ( <param_list> ); ]* }
        behavior {
            [ startup { invoke <method_list> generate <msg_list> } ]
            [ cleanup { invoke <method_list> generate <msg_list> } ]
            [ received <received_msg> {
                invoke <method_list> generate <msg_list> } ]*
        }
        notes {
            [ <note_name> = <note_content> ; ]*
        }
    }
```

그림 3: IDN Syntax

2.3.2 ADN(Architecture Description Notation)

ADN은 IDN으로 정의된 컴포넌트들과 C2 커넥터들 사이의 바인딩으로 구성된 아키텍처 명세언어를 정의한다. ADN 구문은 아키텍처의 컴포넌트와 커넥터의 인스턴스 선언 부분, 그리고 선언된 컴포넌트와 커넥터의 인스턴스들 간의 연결 관계를 나타내는 부분으로 구성되어 있다.

```
<architecture> ::=
    package <package_name>
    [ import <import_list> ]*
    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 4: ADN Syntax

3. ADL 모델을 Java 구현 모델로 매핑하는 방법

본 장에서는 2장에서 설명한 ADL로 기술된 아키텍처 모델을 Java 구현 모델로 매핑하는 방법을 기술한다.

3.1 ADL 모델의 Java 구현 모델

그림 5는 본 논문에서 제안하는 매핑 규칙에 따라 ADL 모델로부터 생성되는 Java 구현 모델을 보여준다.

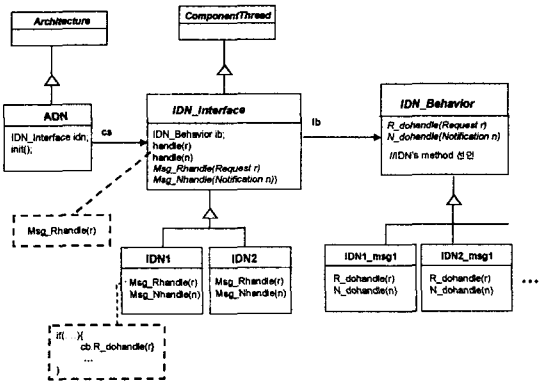

그림 5: ADL 모델의 Java 구현 모델

그림 5에서 ADN 클래스는 C2 Class Framework의 *Architecture* 클래스를 확장하여, ADN으로 기술된 아키텍처 모델을 구현한 클래스이다. 그리고 *IDN_Interface* 클래스와 *IDN_Behavior* 클래스는 *ComponentThread* 클래스를 확장하여, IDN으로 기술된 컴포넌트 모델을 구현한 클래스이다. 이와

같이 IDN으로 기술된 컴포넌트 모델이 *IDN_Interface* 클래스와 *IDN_Behavior* 클래스로 나뉘어 구현됨으로써 메시지 구분 코드, 컴포넌트의 메소드 선언 코드, 메시지 핸들링 코드들이 서로 분리되고 이는 클래스들 사이의 의존도를 줄인다. 모든 컴포넌트 명세 클래스는 *IDN_Interface* 클래스의 서브클래스로 정의한다. 이 서브클래스에서 하는 역할은 " *received message*" 를 구별하는 역할을 정의하는 것이다. ADL을 구성하는 모든 컴포넌트의 메소드는 *IDN_Behavior* 클래스에 정의한다.

3.2 IDN 모델의 Java 코드 매핑 규칙

IDN으로 기술된 컴포넌트 모델을 Java 구현 코드로 변환하는 매핑 규칙은 그림 6과 같다. 그림 6에서 같이, 컴포넌트 명세를 클래스화 할 때, C2 Class Framework의 *Component* 또는 *ComponentThread*를 상속 받아 사용한다. 그리고 컴포넌트 명세에서 정의한 각각의 method를 클래스의 멤버함수로 선언한다. 컴포넌트 명세의 behavior에서 handle(Request r), handle(Notification n)에 대한 정보를 얻고 이것을 자바 코드로 표현한다. 그림 7은 그림 6의 매핑 규칙에 따라 IDN 모델로부터 생성된 Java 골격 코드를 보여준다.

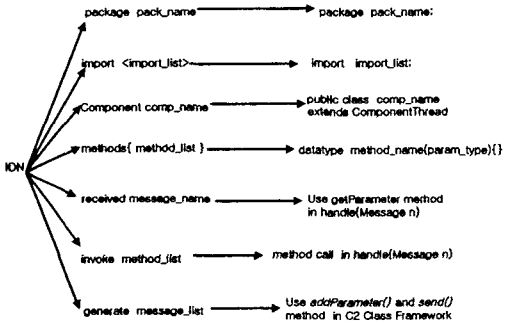

그림 6: IDN 모델의 매핑 규칙

```
package package_name;
import c2.framework.*; ...
public class component_name extends ComponentThread
{
  // 생성자 선언
  public component_name();
  public component_name(String name) { create(name);}
  // 컴포넌트 초기화
  public void create(String name){ ...}
  // 컴포넌트의 메소드 선언
  private method1(int p1) { ... }
// 메시지 핸들링 메소드 정의
  public synchronized void handle(Request r) {
    String msg_name ; int msg_size ;
      msg_name = r.name(); msg_size = r.size();
    if(msg_name.equalsIgnoreCase("msg"){          // received message 구분
      if(size == 1){
          Integer temp_msg = r.getParameter("p1");   // 메시지 정보 얻기
          method1(temp_msg);                         // method call
          //다른 컴포넌트에게 전달할 새로운 메시지 생성
          Notification new_msg = new Notification("bottom");
          p2 = p1; new_msg.addParameter("p2",p2);
        send(new_msg);                               // 메시지 전달
      }
    }
    public synchronized void handle(Notification n){ ...}
  }
}
```

그림 7: IDN 모델의 Java 골격 코드

3.3. ADN 모델의 Java 코드 매핑 규칙

ADN으로 기술된 아키텍처 모델을 Java 구현 코드로 변환하는 매핑 규칙은 그림 8과 같다. 아키텍처 명세는 C2 Class Framework의 *Architecture* 클래스를 상속받아 확장한 Java

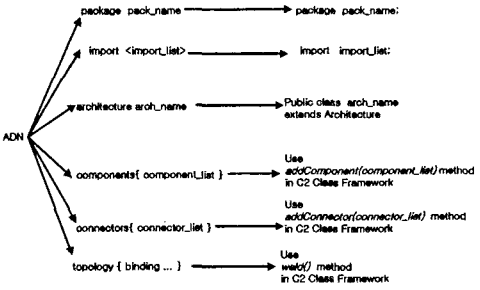

그림 8: ADN 모델의 매핑 규칙

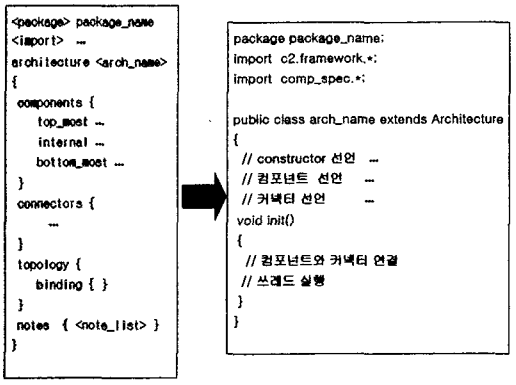

그림 9: ADN 모델의 Java 골격 코드

구현 코드로 매핑된다. 그림 9는 그림8의 매핑 규칙에 따라 ADN 모델로부터 생성된 Java 골격 코드를 보여준다.

4. 결론

본 논문에서는 ADL 모델을 자바 구현 모델로 변환하는 방법을 제시하였다. 본 논문의 방법을 토대로 상위수준에서 정의한 ADL 모델로부터 자바 골격 코드를 자동 생성하는 도구를 구현하였다.

참고 문헌

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

[2] R.N.Talyor, et al., "A Component and Message Based Architectural Style for GUI Software" , IEEE Trans. Software Eng., June 1996, pp. 390-406

[3] UCI, C2 Java and C++ Class Frameworks, http://ftp.ics.uci.edu/pub/arch/software.html

[4] 노성환, 신동익, 최재각, 전태웅, "C2 스타일의 아키텍처 기술을 지원하는 ADL 정의" 한국정보과학회 추계학술발표회 논문집 제28권 2호, 2001년 10월, pp. 370-372.

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