ADL 처리기의 설계와 구현

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

Design and Implementation of an ADL Processor

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

요 약

잘 정의된 소프트웨어 아키텍처는 복잡성과 무형성으로 인한 소프트웨어의 개발과 진화의 어려움을 상당히 해소할 수 있다. 소프트웨어 시스템을 아키텍처 수준에서 효과적으로 설계, 분석하기 위해서는 아키텍처 기술 언어(ADL)와 이를 지원하는 도구의 사용이 필요하다. 본 논문은 컴포넌트 기반의 도메인 아키텍처 모델링 시 C2 스타일의 아키텍처 기술을 지원할 수 있도록 본 연구팀이 정의한 ADL로 기술된 아키텍처 모델의 편집과 구문 및 의미 상의 오류 검사를 지원하는 ADL 처리기의 설계와 구현 방법을 기술한다.

1. 서 론

소프트웨어 기술이 여러 응용 분야에 걸쳐 눈부신 발전을 거듭하고 있다. 하지만 산업체에서 소프트웨어의 생산성과 품질은 여전히 해결해야 할 큰 문제로 남아있다. 그러한 근본적인 원인은 소프트웨어가 갖는 본질적인 복잡성과 무형성(intangibility)에 있다고 볼 수 있으나, 잘 정의된 소프트웨어 아키텍처는 시스템 수준에서의 소프트웨어의 구성 요소, 이들의 상호 관계, 그리고 적용된 설계 사상(design rationale)들을 명시적으로 드러나게 함으로써 복잡성과 무형성으로 인한 소프트웨어의 개발과 진화의 어려움을 상당히 해소할 수 있다.

소프트웨어 아키텍처는 지난 10여년간 해외에서 활발한 연구가 이루어져 현재 소프트웨어를 아키텍처 수준에서 분석, 설계하고 이를 토대로 소프트웨어를 개발하는데 유용한 이론적 기반과 응용 사례들이 많이 축적되고 있다. 소프트웨어 아키텍처를 정확하고 엄밀하게 기술하고 분석하는데 필요한 아키텍처 기술 언어(ADL:Architecture Description Language)들도 다수 소개되어 있다. 국내에서도 아키텍처에 대한 관심과 연구들이 점차 많아지고 있다. 그러나 아직까지 실제 상용 소프트웨어 개발에 ADL을 사용한 사례는 흔치 않다. 소프트웨어 개발에서 아키텍처의 중요성이 점증하고 있음에 비추어 사용이 용이하면서 기술적으로 성숙된 ADL과 이를 실용적으로 지원할 수 있는 도구의 개발이 시급하다.

본 연구팀은 컴포넌트 기반의 도메인 모델링 시 C2 스타일[1]의 아키텍처를 자바와 유사한 구문으로 컴포넌트 명세와 아키텍처 명세를 분리하여 기술할 수 있는 ADL을 정의하였다. 본 논문은 본 연구팀이 정의한 ADL로 기술된 아키텍처 모델의 편집과 구문 및 의미 상의 오류 검사를 지원하는 ADL 처리기의 설계와 구현 방법을 기술한다.

2. C 스타일 기반의 ADL

C2 스타일은 UCI(University of California in Irvine)에서 본래 GUI를 갖는 응용 소프트웨어 개발을 지원하기 위하여 설계한 아키텍처 스타일이다. 본 연구팀은 C2 스타일의 아키텍처를 Java와 유사한 구문으로 컴포넌트 명세와 아키텍처 명세를 분리하여 기술할 수 있는 ADL을 정의하였다. ADL은 UCI의 C2SADL[2]에 대응하는 아래의 2가지 표기형식(IDN, ADN)으로 설계하였다.

① 컴포넌트 명세 언어(IDN:Interface Definition Notation): 포트 기반 인터페이스, 내부 메소드 및 메시지 전이 행위들로 표현되는 컴포넌트 명세언어.

② 아키텍처 명세 언어(ADN:Architecture Definition Notation): IDN으로 정의된 컴포넌트들과 C2 커넥터들 사이의 바인딩들로 구성된 아키텍처 명세 언어.

3. ADL 시스템 구조

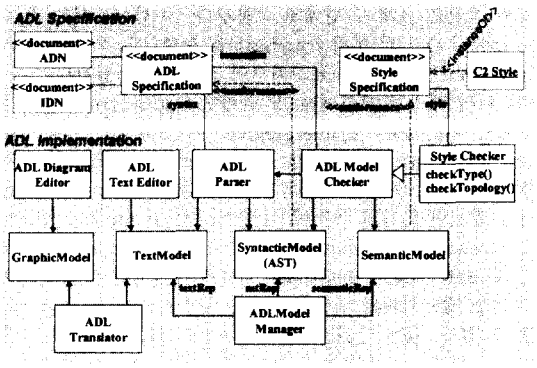

그림 1: Overall Structure of ADL System

ADL 시스템은 (그림1)과 같이 크게 ADL 명세와 ADL 구현으로 구성된다.

ADL 명세는 본 연구팀이 IDN과 ADN으로 분리하여 정의한 ADL의 구문과 의미에 대한 명세이고, 스타일 명세는 아키텍처에 대한 스타일 명세로서 본 연구에서는 C2 스타일 아키텍처의 타입과 형세 규칙에 대한 명세이다.

ADL 시스템의 구현인 ADL 처리기와 ADL 모델들은 외부와 ADL 인터페이스들을 통해서 연결된다. (그림1)에서 ADL 다이어그램 편집기, 그래픽 모델, 그리고 ADL변환기 부분은 본 연구와 병행하여 ETRI에서 별도로 개발 중에 있다.

본 연구팀이 개발하고자 하는 ADL 처리기는 텍스트 형태의 ADL 명세를 편집하고 주어진 ADL 명세에 대한 구문 및 의미 검사를 지원하는 도구이다.

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

4. ADL 처리기

ADL 처리기는 ADL로 기술된 아키텍처 모델을 편집하기 위한 ADL 편집기와 ADL 모델의 구문, 타입, 그리고 형세 검사를 지원하는 ADL 모델검사기로 구성된다.

① ADL편집기 - 아키텍처 설계자가 소프트웨어 시스템의 컴포넌트 합성 구조를 ADL 텍스트 형태로 표현하는 작업을 지원한다.

② ADL모델검사기- 작성된 ADL 모델이 ADL의 구문, 스타일 명세을 준수하는지 여부를 검사한다.

본 연구는 ETRI에서 개발 중인 C2 스타일 기반의 컴포넌트 조립 지원도구에서 텍스트 또는 그래픽 형태로 기술한 아키텍처 명세 또는 컴포넌트 명세에 대한 구문과 스타일 검사를 지원하는 도구를 개발하는 것이다. 따라서 본 연구에서 개발하는 ADL 처리기는 ETRI에서 개발 중인 컴포넌트 조립 지원도구의 다른 모듈들과 통합이 가능한 형태로 설계하였다.

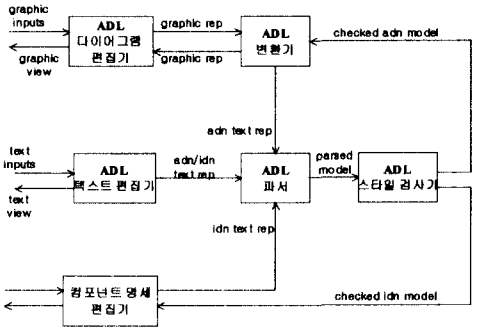

그림 2: Data flow of ADL Processor

(그림2)는 ADL 처리기의 데이터 흐름을 나타낸다.

ADL 다이어그램 편집기를 통해 편집된 아키텍처 다이어그램은 텍스트 형태의 ADN 모델 명세로 변환된 후 ADL 처리기에 의해 구문 및 스타일 검사가 수행된다. ADL 변환기는 ADL 다이어그램 편집기를 통해 편집된 그래픽 형태의 아키텍처 다이어그램을 텍스트 형태의 ADN으로 변환하는데 필요한 모델 요소 정보를 획득, 설정하거나 스타일 검사를 마친 텍스트 형태의 ADN 모델 명세를 이에 대응하는 아키텍처 다이어그램으로 변환하는데 필요한 모델 요소 정보를 획득, 설정한다.

컴포넌트 명세 편집기는 ADL 편집기와 달리 위저드(wizard) 형태로 컴포넌트 명세를 편집하는 도구이다. 컴포넌트 명세 편집기 상의 컴포넌트 명세에 대한 구문 및 모델 검사도 ADL 처리기의 ADL 파서와 ADL 스타일 검사기를 통해 이루어진다.

4.1 ADL 편집기

ADL 편집기는 ADL 텍스트 모델을 편집하기 위한 것으로서 MDI(Multiple Documentation Interface) 환경에서 기본적인 편집기의 기능을 제공한다.

ADL 편집기에는 MVC(Model-View-Controller) 모델을 적용했다.

(그림 3)은 ADL 편집기에 대한 클래스 다이어그램이다.

(그림3)에서 다수의 ADL 텍스트 모델을 관리하는 ADLEditor 클래스, 하나의 텍스트 모델과 관련된 요소들을 관리하는 ADLTextModel 클래스, 텍스트 모델을 나타내는 ADLDocument 클래스, 텍스트 모델 편집과 관련된 상태 정보를 갖는 ADLTextModelStatus 클래스들은 MVC의 Model에 해당된다 ADL 텍스트 모델을 화면상에 보여주는 ADLTextView 클래스와 메뉴 관련 클래스들은 MVC의 View에 해당된다. 그리고 메뉴와 관련된 액션들은 MVC의 Controller에 해당된다.

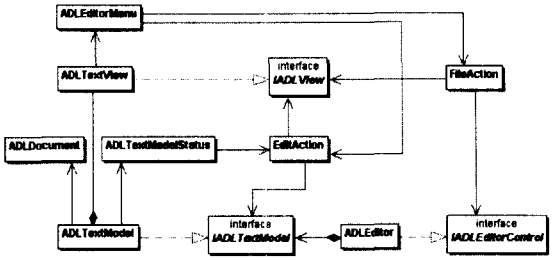

그림 3: ADL 편집기의 클래스 다이어그램

4.2 ADL 모델검사기

ADL 모델에 대한 구문과 C2 아키텍처 스타일 명세(타입 및 형세 규칙)의 준수 여부를 검사한다.

(그림4)은 ADL모델검사기를 구성하는 인터페이스와 클래스를 표현한 다이어그램이다.

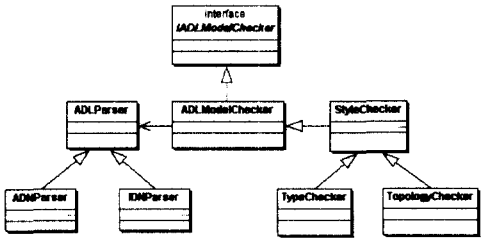

그림 4: ADL 모델검사기의 클래스 다이어그램

*IADLModelChecker*는 ADL 검사기에서 외부 모듈에게 제공하는 인터페이스이다. 외부 모듈은 *IADLModelChecker* 인터페이스만을 통해 ADL검사기에서 제공하는 검사 기능을 이용할 수 있다.

ADL 처리기의 외부 모듈은 *IADLModelChecker* 인터페이스 타입의 객체를 통해 ADL 모델의 구문, 타입, 그리고 형세 검사에 대응하는 메소드를 호출하여 ADL모델검사기의 해당 검사 기능을 이용한다.

(그림5)는 외부 모듈이 ADL 모델 검사기를 이용하여 구문, 타입, 및 형세 검사를 수행하는 과정을 대략적으로 보여주는 순서도이다.

외부 모듈로부터 *IADLModelChecker*를 통해 구문 검사 요청이 들어 오면 ADL 파서는 ADL 텍스트모델을 입력 받아 구문 분석하여 ADL 구문모델(syntactic model)을 생성한다. (그림5)에서 볼 수 있듯이 ADL 텍스트 모델의 구문 검사는 타입 검사의 선행 검사이고, 타입 검사는 형세 검사의 선행 검사이다. 따라서 외부모듈로부터 타입검사 요청이 들어왔을 때 ADLModelChecker 객체는 구문 검사가 선행되었는지를 점검하여 만약 구문검사가 선행되지 않았다면 구문검사를 우선 수행한 후 타입검사를 수행한다. 형세검사도 타입검사와 동일한 방법으로 타입검사가 선행되었는지를 우선 점검한다. 타입 검사는 성공적인 구문 검사 결과로 생성된 ADL 구문모델을 이용하여 이에 대응하는 ADL 시맨틱모델(semantic model) 생성한 후 타입 검사를 수행한다. 형세 검사는 타입 검사 시 생성된 ADL 시맨틱 모델을 이용하여 형세 검사를 수행한다. (그림5)에서 생략되었지만 ADL 모델의 구문, 스타일 검사 중 오류가 발생하면 외부 모듈이 검사 요청 시 파라미터로 전달 받은 오류 메시지 리스트에 오류 정보를 담아 외부 모듈에 알려준다.

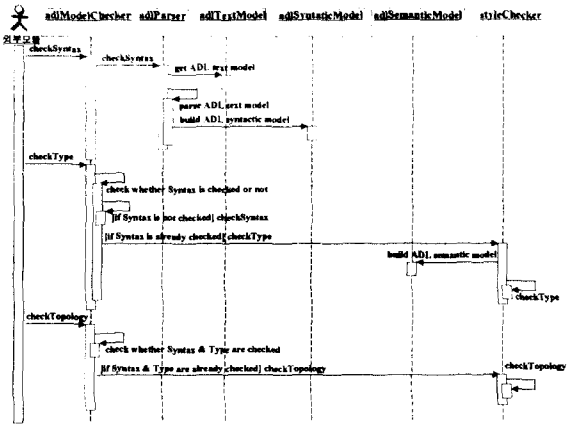

그림 5: ADL 모델에 대한 구문, 타입 및 형세 검사 과정

4.2.1 ADL 파서

본 연구팀이 정의한 ADL이 IDN과 ADN으로 분리되어 있기 때문에 ADL 파서는 컴포넌트 명세를 위한 IDN 파서, 아키텍처 명세를 위한 ADN 파서로 분리된다.

외부모듈이 IADLModelChecker 인터페이스 통해 요청한 구문 검사의 대상이 컴포넌트 명세인지 아키텍처 명세인지에 따라 ADLModelChecker 클래스는 IDNParser 또는 ADNParser에게 구문 검사 요청을 한다.

(그림6)는 IDN파서의 클래스 다이어그램이다. IDNParser의 구문 검사 기능은 Parser 클래스의 해당 메소드에 구현된다. Parser 클래스는 Scanner 클래스로부터 토큰(token)화 된 IDN 명세를 제공받아 구문 분석을 하여 궁극적으로 파스트리(parse tree)를 생성하는 순환적 내림차순(recursive-descent) 파서이다. Checker 클래스는 Parser 클래스가 생성한 파스트리를 이용하여 기본적인 컨텍스트 분석(contextual analysis)을 수행한다. Modeler 클래스는 기본적인 컨텍스트 분석을 마친 파스트리를 이용하여 IDN 모델의 요소 정보를 포함하고 있는 IDNModel 객체를 생성한다. 생성된 IDNModel 객체는 타입 및 형세 검사를 위하여 타입검사기에 의해 시맨틱(semantic) 모델로 변환된다.

아키텍처 명세에 대한 구문 검사를 수행하는 ADN파서의 클래스 구조와 수행 과정도 IDN 파서와 유사하다.

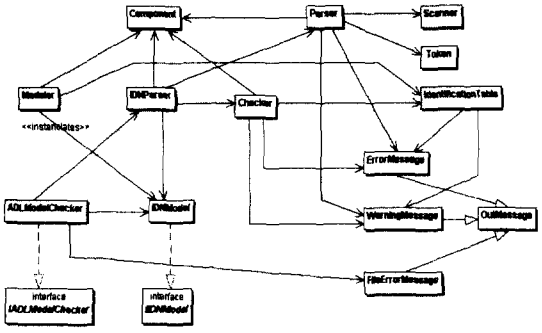

그림 6: IDN파서의 클래스 다이어그램

4.2.2 ADL 스타일 검사기

ADL 스타일 검사는 아키텍처 명세에 대한 타입 검사와 형세 검사이다.

본 연구에서의 타입 검사는 아키텍처 상에서 송신 컴포넌트의 송신 메시지 집합과 수신 컴포넌트의 수신 메시지 집합 간의 포함 관계를 검사하는 것을 의미한다. 형세 검사기는 ADL 모델 명세가 형세 규칙과 제약 조건을 준수하는지를 검사한다. 타입검사는 형세 검사의 선행 검사이다.

ADL 스타일 검사기는 ADL로 기술된 아키텍처 명세의 시맨틱 모델을 ADL 파서에 의해 생성된 파스트리를 토대로 생성한 후 타입 오류와 형세 오류를 검사한다.

아키텍처 명세의 파스트리로부터 시맨틱 모델을 생성하는 과정은 Builder 패턴[3]을, 그리고 생성된 아키텍처 명세의 시맨틱 모델에 대한 타입 및 형세 검사 과정에는 Visitor 패턴[3]을 적용하여 설계한다. 또한 ADL 모델 명세의 타입과 형세에 대한 규칙 및 제약 조건들은 스타일 검사 코드나 검사 대상 시맨틱 모델과 별도로 구성된다. 이러한 방식으로 설계된 스타일 검사기는 모델 검사 기능의 변경이나 추가가 용이하게 된다.

5. 결론 및 향후 연구

본 논문에서는 C2 스타일의 아키텍처 기술을 지원하는 ADL 처리기의 설계와 구현 방법을 기술하였다. 본 ADL 처리기는 ETRI에서 개발하고 있는 아키텍처 기반의 컴포넌트 조립 지원도구에서 ADL 지원 도구로서 사용될 예정이다. 따라서 ADL 처리기 설계 시 컴포넌트 조립 지원 도구의 다른 모듈들과의 상호 작용을 고려하여 설계하였고, 향후 추가적으로 요구되는 기능들을 쉽게 지원할 수 있도록 설계하였다. 그리고 현재 ADL 처리기의 ADL편집기와 ADL파서는 구현되어 ADL 처리기의 외부 모듈들과 통합 시험 중에 있다. ADL 스타일 검사기는 개발 중에 있다.

향 후 연구로서, 본 논문의 ADL 처리기를 다음과 같은 도구들로 확장할 계획이다.

- 정적, 동적인 분석 도구
- 정제(refinement) 도구 - 코드 생성기, UML 변환기 등
- 테스트 도구 - 테스트케이스 생성, 자동 테스트 오러클 등

참고문헌

[1] 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

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

[3] Erich Gamma, et al., "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1994.

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

[5] W. F. Tichy, "Software Development Control Based on Module Interconnection", Proceedings of the 4th Int'l Conference on Software Engineering, 1979, pp. 29-41

[6] R. Prieto-Diaz and J. M. Neighbors, "Module Interconnection Languages", The Journal of Systems and Software, Vol. 6, No. 4, November 1986, pp. 307-334

[7] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide," Addison-Wesley, 1999

[8] David A. Watt and Deryck F. Brown, "Programming Language Processors in Java," Prentice Hall, 2000.