C2 스타일의 아키텍쳐 기술을 지원하는 ADL 지원도구의 개발

신 동 익[†]·노 성 환[†]·최 재 각[†]·전 태 웅[††]
이 승 연[†††]·권 오 천[††††]·신 규 상[†††††]

요 약

최근, 소프트웨어의 재사용성, 생산성, 그리고 품질을 높이기 위한 방법으로서 컴포넌트 기반의 소프트웨어 개발(CBD : Component-Based Development) 방식이 빠르게 확산되고 있다. CBD를 효과적으로 수행하기 위해서는 응용 컴포넌트들이 서로 정확하게 결합하여 작동할 수 있는 아키텍쳐를 기반으로 하여 컴포넌트의 생성과 합성 작업이 이루어질 수 있어야 한다. 소프트웨어 아키텍쳐는 아키텍쳐 기술 언어(ADL : Architecture Description Language)를 사용하여 기술되어야 정확하고 엄밀한 아키텍쳐 모델링이 가능하다. 본 논문에서는 도메인 아키텍쳐 기반의 CBD에 효과적으로 사용될 수 있는 ADL 지원도구의 시스템 아키텍쳐를 제안하고, 제안한 시스템 아키텍쳐의 각 구성 요소들에 대하여 기술한다. 그리고 C2 스타일의 아키텍쳐 기술을 지원하는 UCI(University of California in Irvine)의 C2SADL을 변경하여 재정의한 ADL과 지원도구로서 개발 중인 ADL 지원도구의 설계 및 구현 방법을 기술한다. 본 연구팀이 개발 중인 ADL 지원도구는 본 논문에서 제안한 ADL 지원도구의 시스템 아키텍쳐의 일부 구성 요소들을 구현한다.

Development of an ADL tool set that supports the description of C2-style architecture

Dong-ik Shin[†]·Sung-hwan Roh[†]·Jae-kak Choi[†]·Taewoong Jeon[††]
Seung-yun Lee[†††]·Oh-cheon Kwon[††††]·Gyu-sang Shin[†††††]

ABSTRACT

Recently, component-based development (CBD) is rapidly spreading as a way of improving the reusability, productivity, and quality of software. For CBD to be effective in achieving such design objectives, the creation and integration of components must be based on a well-defined architecture that guides the correct composition and cooperation of application components. Software architecture must be described using an architecture description language (ADL) to ensure the correctness and preciseness of architecture models. In this paper, we propose the system architecture of an ADL tool set that can effectively support the use of CBD based on the domain architecture and we describe each component of the proposed system architecture. We also modify and redefine C2SADL that was developed to support the use of the description of C2 architectural style by UCI (University of California in Irvine) to facilitate the integration of separately described architecture models, and introduce the method of design and implementation of our ADL processor that partially implements the proposed ADL system architecture.

키워드 : 컴포넌트(Component), 컴포넌트 기반의 개발(Component-Based Development(CBD)), 소프트웨어 아키텍쳐(Software Architecture), 아키텍쳐 기술 언어(Architecture Description Language), C2 아키텍쳐 스타일(C2 Architectural Style), 도메인 모델링(Domain Modeling)

1. 서 론

최근, 소프트웨어의 재사용성, 생산성 그리고 품질을 높이기 위한 방법으로서 컴포넌트 기반의 소프트웨어 개발(CBD : Component-Based Development) 방식이 빠르게 확산되고 있다. CBD는 컴포넌트의 개발 뿐만 아니라 컴포넌트들의 합성에 의한 시스템 개발을 포함한다.

CBD를 효과적으로 지원하기 위해서는 응용 컴포넌트들이 서로 정확하게 결합하여 작동할 수 있는 아키텍쳐를 기반으로 컴포넌트의 생성과 합성 작업이 이루어질 수 있어야 한다. 아키텍쳐 기반 소프트웨어 개발은 상위 수준의 소프트웨어 시스템 모델들이 소프트웨어에 대한 분해, 정제, 구현, 합성 및 진화의 방향을 지시하는 개발 방식이다. 잘 확립된 소프트웨어 아키텍쳐는 소프트웨어를 컴포넌트 단위

† 준 회 원 : 고려대학교 대학원 전산학과
†† 정 회 원 : 고려대학교 전산학과 교수
††† 정 회 원 : 한국전자통신연구원 컴포넌트공학 연구팀 연구원
†††† 정 회 원 : 한국전자통신연구원 컴포넌트공학 연구팀 책임연구원
††††† 정 회 원 : 한국전자통신연구원 컴포넌트공학 연구팀장
논문접수 : 2001년 10월 4일, 심사완료 : 2001년 12월 21일

로 분해, 생성, 합성, 개조를 가능하게 함으로써 복잡 방대하고, 다양한 플랫폼과 장기간 운영을 지원해야 하는 성격의 시스템들에 대한 개발과 진화를 상당히 개선할 수 있다.

소프트웨어 아키텍처는 아키텍처 기술 언어(ADL : Architecture Description Language)로 기술되어야 정확하고 엄밀하게 아키텍처를 모델링할 수 있고 아키텍처를 기반한 시스템 분석, 정제 검증이 가능하다. ADL과 지원도구에 대한 연구들은 90년대 초반부터 외국에서 활발히 진행되어 왔다[1,2]. 그러나 현존하는 대부분의 ADL 지원도구들은 프로토타입 수준에 머물러 있어 ADL이 상용 소프트웨어의 개발에 실제 사용된 예는 흔치 않다. 소프트웨어를 아키텍처 수준에서 정확하고 엄밀하게 설계, 분석하는 것이 점차 중요해지고 있음에 따라 사용이 용이하면서 기술적으로 성숙된 ADL과 지원도구의 실용화가 요구된다.

본 논문에서는 도메인 아키텍처 기반의 CBD에 효과적으로 사용될 수 있는 ADL 지원도구의 시스템 아키텍처를 제안하고, 제안한 시스템 아키텍처의 각 구성 요소들에 대하여 기술한다. 그리고 UCI(University of California in Irvine)의 C2 스타일[4]의 아키텍처 기술을 지원하는 C2SADL[18]을 변경하여 재정의한 본 연구팀의 ADL과 이의 지원도구로서 개발 중인 ADL 지원도구의 설계 및 구현 방법을 기술한다. 본 연구팀이 개발 중인 ADL 지원도구는 본 논문에서 제안한 ADL 지원도구의 시스템 아키텍처의 일부 구성 요소들을 구현한다.

본 논문의 구성은 다음과 같다.

2장에서는 ADL과 지원도구와 관련한 연구 사례들을 살펴보고 3장에서는 도메인 아키텍처 기반의 응용 컴포넌트들의 효율적인 생성과 합성에 효과적으로 사용될 수 있는 ADL에 요구되는 표현 능력들과 그러한 ADL로 기술되는 아키텍처의 메타 모델을 제시한다. 4장에서는 ADL 지원도구의 핵심 기능들에 대하여 살펴본다. 5장에서는 ADL 지원도구의 전체적인 시스템 구성과 각 서브 시스템들에 대하여 살펴본다[3]. 그리고 6장에서는 본 연구팀의 ADL과 개발 중인 지원도구의 설계 및 구현 방법에 대하여 소개한다. 마지막으로 7장에서는 결론 및 향후 연구 내용에 대하여 기술한다.

2. 관련 연구

본 장에서는 종래에 소프트웨어 아키텍처를 기술하기 위하여 사용되었던 표기 형식들과 이들의 한계점을 간략하게 설명하고, 현재 나와 있는 ADL들의 유형별 특징을 간략하게 기술한다.

종래에 소프트웨어 아키텍처를 기술하기 위하여 많이 사용되었던 표기 형식들은 비정형적인 블록 다이어그램, 프로그래밍 언어에서 제공하는 MIL(Module Interface Language) 또는 IDL(Interface Definition Language)과 같은 컴포넌트 인터페이스 정의 언어[5, 6], 그리고 UML[7]의 클래스/컴포넌트 다이어그램들이다. 이러한 종래의 아키텍처 표기 형식들은 아키텍처의 구성 요소들과 이들의 연결 관계를 엄밀하게 표현하지 못하거나(비정형 블록 다이어그램) 컴포넌트들의 정의/사용(definition/use) 관계 또는 인터페이스의 import/export 관계를 주로 표현하여 아키텍처의 높은 추상화 수준의 상호 관계를 표현하지 못한다.

위와 같은 종래의 아키텍처 기술 방식들이 갖는 한계를 개선하기 위해 90년대 초반부터 아키텍처를 명시적이고 정확하게 기술할 수 있는 아키텍처 기술 언어(ADL)와 ADL 지원 환경에 대한 연구가 외국에서 활발히 진행되어 왔다. 그 결과, 현재 다양한 ADL들과 지원도구들이 소개되어 있다. 지금까지 나와있는 ADL들을 유형 별로 요약하면, 1) 시맨틱 모델 기반 ADL 2) 스타일 기반 ADL 3) 도메인 기반 ADL 4) ADL 인터체인지(interchange) 언어들로 구분할 수 있다.

시맨틱 모델 기반 ADL은 아키텍처의 구조적 또는 행위적 성질들을 정형 시맨틱 모델에 의거하여 엄밀하게 기술하는 표현 수단을 제공하며 Wright[8], Rapide[9], Darwin[10] 등이 이에 속한다. 스타일 기반 ADL은 특정한 아키텍처 스타일을 따르는 아키텍처를 기술하거나 아키텍처 스타일에 대한 기술을 지원하며 Unicon[11], Aesop[12], C2SADL[13] 등이 이에 속한다. 도메인 기반 ADL은 특정 도메인에 속한 아키텍처들의 기술에 적합한 언어로서 해당 도메인에서 중요한 아키텍처 측면들을 잘 기술할 수 있다. 예를 들면, ROOM[13]과 MetaH[15]는 실시간 또는 내장 시스템들의 아키텍처들을 기술하는데 적합한 ADL들이다. ADL 인터체인지 언어[16]는 다수의 ADL들을 병용하여 사용할 수 있도록 서로 다른 ADL로 작성된 아키텍처 기술(architecture description)들의 상호 변환과 공유를 지원하며 Acme[17]가 이에 속한다.

현존의 ADL들은 대부분 외국의 대학이나 연구기관에서 연구 프로젝트의 수행 결과로 얻어진 것들이다. 이들의 지원도구들도 함께 제작되어 있거나 계속 개발이 진행 중에 있지만 아직은 대부분 프로토타입 수준에 머물러 있다. 따라서 산업체에서 실제 상용 소프트웨어의 개발에 ADL이 사용된 예는 흔치 않다. 그러나 소프트웨어 아키텍처 기술에 ADL의 사용을 시험적으로 적용한 사례들은 많다. 국내에서는 ADL이 개발되거나 사용된 사례가 아직 없다.

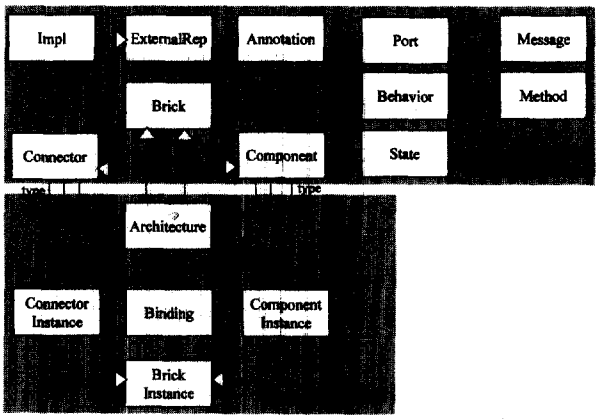
기존의 ADL과 지원도구들에 대한 분류와 비교는 [2]에 기술되어 있다. 본 연구팀의 ADL 지원도구는 기존 ADL 지원도구와 달리 EJB 컴포넌트 기반의 응용 소프트웨어 개발을 지원하기 위한 도구이다.

3. 효과적인 도메인 아키텍처 모델링을 위한 ADL

본 장에서는 컴포넌트 기반의 도메인 아키텍처 모델링

시, 아키텍쳐 수준에서 소프트웨어를 정확하고 엄밀하게 모델링하고 아키텍쳐 기반의 시스템 분석, 정제, 및 검증을 지원할 수 있는 ADL에 요구되는 표현 능력과 그러한 ADL로 기술되는 아키텍쳐의 메타모델을 제시한다.

아키텍쳐 수준에서의 컴포넌트들은 높은 추상화 수준의 상호관계를 갖기 때문에 아키텍쳐의 효과적인 설계 및 분석을 위해서는 컴포넌트들과 동등한 수준에서의 커넥터들을 직접적으로 표현할 수 있어야 한다. 이를 위해 (그림 1)과 같이 커넥터도 컴포넌트와 동등한 수준에서 행위와 구조를 기술 할 수 있어야 한다.



(그림 1) 효과적인 ADL로 기술된 아키텍쳐의 메타모델

효과적인 도메인 아키텍쳐 모델링을 위하여 컴포넌트와 커넥터 모델의 기술에 요구되는 ADL의 표현 능력은 다음과 같다.

- 컴포넌트와 커넥터의 인터페이스는 포트들로 표현된다.
- 컴포넌트들과 커넥터들의 인터페이스인 포트들로 연결되어 구성된 아키텍쳐의 컴포넌트들 간의 상호 작용은 커넥터를 통해 포트들로 연결된 컴포넌트들 간의 메시지(message) 송수신으로 표현한다.
- ADL은 컴포넌트 또는 커넥터의 동작 시작, 실행, 또는 종료 시 행해지는 행위를 메소드 호출과 포트를 통한 메시지 송신으로 표현한다. 또한 컴포넌트 또는 커넥터의 상태(state)에 종속적인 행위를 표현한다.
- 컴포넌트 또는 커넥터는 서브 아키텍쳐를 가질 수 있다. 즉, ADL은 합성 컴포넌트(composite component)와 합성 커넥터(composite connector)를 표현한다.
- ADL은 컴포넌트들 또는 커넥터들 간의 서브타이핑(sub-typing) 관계를 표현한다.
- ADL이 직접적으로 지원하지 않는 컴포넌트, 커넥터, 또는 아키텍쳐의 속성(property)들은 외부 분석 도구들에 의해 처리될 수 있도록 주석(annotation)으로 표현한다.

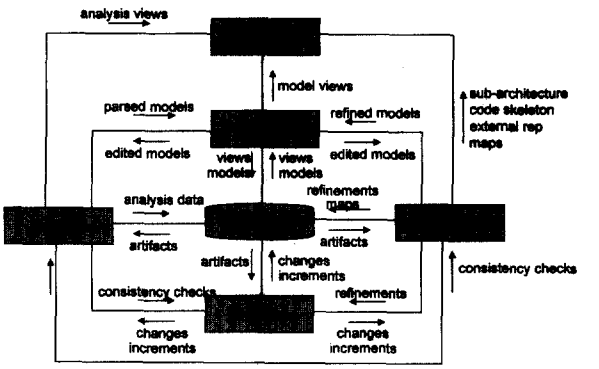
아키텍쳐 모델은 컴포넌트 인스턴스와 커넥터 인스턴스들 간의 바인딩(binding)으로 표현된다 컴포넌트 인스턴스와 커넥터 인스턴스의 타입은 컴포넌트 모델과 커넥터 모델에 정의된다.

(그림 1)은 위에서 기술한 컴포넌트/커넥터 모델 그리고 아키텍쳐 모델에 요구되는 표현 능력을 지원하는 ADL로 기술되는 아키텍쳐의 메타모델이다.

4. ADL 지원도구의 핵심기능

본 장에서는 도메인 아키텍쳐 기반의 응용 컴포넌트들의 효율적 생성과 합성에 효과적으로 사용될 수 있는 ADL 지원도구의 핵심적인 기능들을 기술한다.

(그림 2)는 ADL 지원도구가 아키텍쳐 기반의 CBD를 효과적으로 지원하기 위해 제공해야 할 핵심적인 기능들과 각 기능들간의 데이터 흐름을 나타낸다.



(그림 2) ADL 지원도구의 핵심 기능들

4.1 모델링(Modeling)

ADL 지원도구는 ADL을 사용하여 아키텍쳐를 기술하는데 필요한 편집 기능을 제공한다. 편집은 아키텍쳐 모델의 구성 요소들의 생성, 연결, 수정, 삭제 및 저장 작업들의 지원을 포함한다. 또한 편집된 아키텍쳐 모델이 ADL의 문법에 맞게 기술되었는지 확인할 수 있도록 구문 분석(syntax analysis) 기능을 제공한다. 그리고 ADL의 문법에서 제공되지는 않지만 응용 도메인에 속한 아키텍쳐에 대한 제반 특성들을 기술하기 위한, 해석이 유보된 표현 형식(uninterpreted expressions)의 입력을 허용한다.

4.2 분석(Analysis)

ADL 지원도구는 ADL로 기술된 아키텍쳐 모델에 대한 여러 가지 분석 작업들의 수행을 지원한다. 아키텍쳐 분석은 아키텍쳐 모델에 기술된 내용에 대한 구문적 정확성(syntactic correctness), 의미적 정확성(semantic correctness), 완전성(completeness), 상호 일관성(consistency) 등의 분석을 포함한다. 아키텍쳐 분석은 또한 아키텍쳐 수준에서의 타입 매칭(type matching)과 같은 구조적 특성들, 그리고 기능이나 성능과 같은 행위적 특성들을 찾아내고

이들이 요구된 수준들을 만족하는지 검증한다. 또한 실제로 구현된 시스템과 아키텍처 모델과의 일치성도 분석한다.

4.3 정제(Refinement)

아키텍처 모델은 점차 더 상세한 수준의 모델들로 정제되어 궁극적으로 구현 모델로 변환된다. 정확하고 일관성 있는 아키텍처 모델의 정제를 지원하기 위하여 ADL 지원도구는 서로 다른 상세 수준의 아키텍처 모델들 및 구현 모델들 사이의 매핑(abstraction mapping)이 정확하게 이루어지도록 하고 이러한 매핑 관계를 지속적으로 유지한다. 또한 구현을 지원하기 위하여 ADL 지원도구는 아키텍처 모델로부터 골격 코드(code skeleton)나 프로토타입 코드(prototype code)와 같은 소스코드를 생성한다

4.4 진화(Evolution)

소프트웨어 시스템은 개발 도중이나 개발이 완료된 후에도 지속적으로 변경, 개선되며 이에 따라 소프트웨어 아키텍처도 변경된다. 소프트웨어 아키텍처는 이를 구성하고 있는 컴포넌트들과 커넥터들에 대한 개별적인 변경은 물론, 이들이 합성된 시스템 구성 상의 변경이 필요할 수 있다. ADL 지원도구는 변경에 따른 아키텍처 모델의 개조, 확장, 진화 및 분화를 일정한 규칙에 따라 제어하고 추적할 수 있도록 하여 아키텍처의 변경 작업들이 효율적이고 일관성 있게 이루어질 수 있게 한다.

4.5 시각화(Visualization)

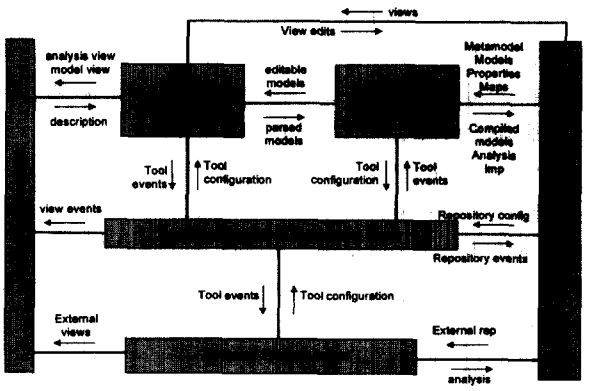
시스템 수준에서의 소프트웨어의 구조적, 행위적 성질들은 매우 다양하고 관여한 사람들의 유형에 따라 관심의 대상이 달라진다. ADL 지원도구는 ADL로 기술된 아키텍처 모델들과 이들의 분석 결과들을 관심의 대상에 따라 이에 적합한 형태로 묘사된 여러 가지 뷰(view)들로 보여준다. 이와 동시에 상호 관련 있는, 서로 다른 뷰에 속한 시스템 성질들 간의 상호 불일치들을 방지 또는 탐지할 수 있는 표현 및 분석 수단을 제공하여야 한다.

5. ADL 지원도구의 시스템 아키텍처

본 장에서는 앞에서 설명한 ADL 지원도구의 핵심 기능들을 지원하기 위해 본 논문에서 제안한 ADL 지원도구의 시스템 아키텍처와 이를 구성하는 각 서브 시스템들에 대하여 기술한다.

(그림 3)는 본 논문에서 제안한 ADL 지원도구의 시스템 아키텍처를 나타낸다.

제안한 ADL 지원도구는 (그림 3)에서와 같이 6개의 서브 시스템들로 구성되어 있다. 서브 시스템들은 2개의 방향에 대해 서비스의 추상화 수준 별로 구분된 계층들로 구성된 다층 구조의 아키텍처를 형성한다.

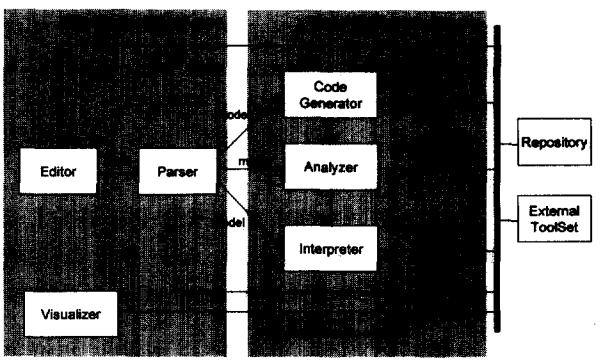


(그림 3) ADL 지원도구의 시스템 구조

각 서브 시스템, 즉 시스템 수준에서의 ADL 지원도구의 각 계층들의 기능은 다음과 같다.

- 사용자 인터페이스 계층(User Interface Layer) : 아키텍처 모델과 분석 데이터를 사용자에게 보여주고 사용자의 입력을 처리한다.
- 뷰 조작 계층(View Manipulation Layer) : 사용자가 이해하는 표기 형식으로 기술된 아키텍처 모델들의 생성, 편집 및 구문 분석을 지원한다.
- 모델 조작 계층(Model Manipulation Layer) : ADL 지원도구가 처리하기 쉬운 내부 형태(internal representation)로 변환된 아키텍처 모델들의 조작, 분석, 변환을 지원한다.
- 툴 조정 및 통합 계층(Tool Coordination & Integration Layer) : ADL 지원도구를 구성하는 툴셋(tool set)의 조정과 통합을 담당한다.
- 외부 툴셋 계층(External Toolset Layer) : ADL이 직접적으로 지원하지 않는 표현들을 이해하고 처리할 수 있는 외부 분석 도구들이다.
- 모델 데이터 저장소 계층(Repository Layer) : 아키텍처 모델과 이와 관련한 가공 데이터(artifacts), 그리고 분석 데이터를 저장, 관리한다.

5.1 뷰/모델 조작 계층(View/Model Manipulation Layers)



(그림 4) 뷰/모델 툴셋 계층들

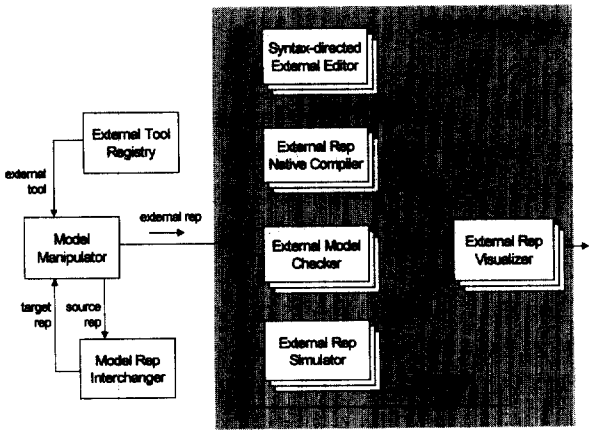
아키텍쳐 모델은 뷰 조작 툴셋과 모델 조작 툴셋에 의해 편집, 컴파일, 분석, 변환된다. 뷰 조작 계층에 속한 툴셋은 사용자가 이해하는 ADL로 기술된 형태의 아키텍쳐 모델에 대한 조작과 시각화(visualization)를 지원한다. 뷰 조작 계층에 의해 편집, 구문 분석된 아키텍쳐 모델은 모델 조작 계층에 속한 툴셋에 의해 분석, 해석, 코드 생성 작업이 이루어진다. (그림 4)는 이와 같은 뷰/모델 조작 계층에 속한 툴셋의 구성 요소와 이들 간의 자료 흐름을 보여준다.

뷰/모델 조작 계층의 각 구성 요소와 기능은 다음과 같다.

- 편집기(Editor) : ADL 표기 형식으로 기술되는 아키텍쳐를 편집한다.
- 파서(Parser) : ADL로 기술된 아키텍쳐 모델에 대한 구문 분석을 수행한다.
- 시각화 출력기(Visualizer) : 아키텍쳐 모델과 분석 데이터를 시각화하여 출력한다.
- 분석기(Analyzer) : 아키텍쳐 모델의 구조적, 행위적 성질들을 분석한다.
- 코드 생성기(Code Generator) : 아키텍쳐 모델로부터 골격 코드(code skeleton)나 프로토타입 코드(prototype code)를 생성한다.
- 해석기(Interpreter) : 아키텍쳐 모델을 해석, 시뮬레이션(simulation)한다.

5.2 외부 툴셋 계층(External Toolset Layer)

아키텍쳐에 대한 분석은 응용 도메인, 아키텍쳐에 적용된 스타일, 분석 목적 등에 따라 분석 작업의 성격과 분석 방법이 크게 달라진다. 따라서 모든 종류의 분석에 필요한 정보가 모두 표현 가능한 구문과 의미를 제공하는 ADL은 현실적으로 존재하기 어렵다. 이와 같은 이유로 인하여 아키텍쳐 분석을 위해 ADL 지원도구는 독립적으로 개발된 외부의 다양한 구현 및 분석 도구들을 필요에 따라 수용할 수 있도록 개방형 아키텍쳐를 가져야 한다.



(그림 5) 외부 툴셋 계층

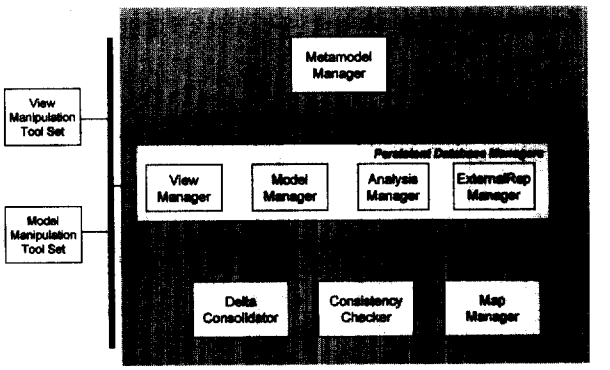
(그림 5)는 본 논문에서 제안한 ADL 지원도구에 통합, 사용되는 외부 도구들의 구성 요소와 이들과 ADL 지원도구 사이의 자료 흐름을 보여준다.

외부 툴셋 계층의 각 구성 요소와 기능은 다음과 같다.

- 외부 편집기(External Editor) : ADL이 지원하지 않는 표현들의 편집을 지원한다.
- 외부 컴파일러(External Compiler) : ADL이 지원하지 않는 표현들의 컴파일을 수행한다.
- 외부 모델검사기(External Model Checker) : ADL이 지원하지 않는 표현들의 모델 분석을 수행한다.
- 외부 시뮬레이터(External Simulator) : ADL이 지원하지 않는 표현들에 대한 의미를 해석하고 시뮬레이션한다.
- 외부 시각화 출력기(External Visualizer) : ADL이 지원하지 않는 표현들을 사용자에게 유용한 형태로 변환하여 출력한다.

5.3 모델 데이터 저장소 계층(Repository Layer)

모델 데이터 저장소 계층은 아키텍쳐 모델링 및 분석 과정 중에 생성된 가공물(artifact)들을 데이터베이스에 저장하고 ADL 지원도구가 분석, 변경, 정제, 변환, 출력 등의 작업을 위하여 해당 가공물을 필요로 하는 경우 이에 대한 검색을 지원한다. 그리고 저장된 가공물 간의 무결성(integrity)을 유지, 보장하고 제약 조건의 위반 시 이를 감지, 처리한다.



(그림 6) 모델 데이터 저장소 계층

모델 데이터 저장소 계층의 각 구성 요소와 기능은 다음과 같다.

- 메타모델 관리기(Metamodel Manager) : ADL로 기술된 아키텍쳐 모델들에 대한 메타모델(metamodel)을 관리한다.
- 지속성 DB 관리기(Persistent DB Manager) : 아키텍쳐 모델들의 뷰(view), 모델(model), 분석(analysis), 그리고 ADL이 지원하지 않는 표현들을 데이터베이스에

저장, 관리한다.

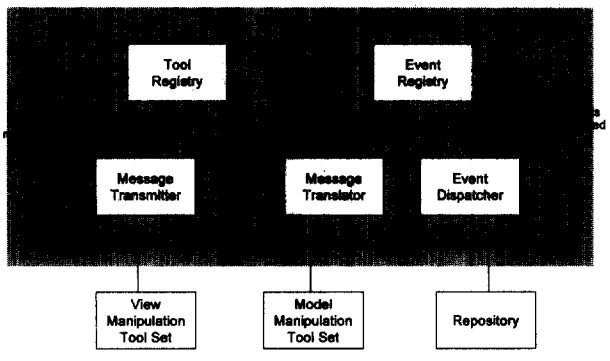
- 모델 매핑 관리기(Map Manager) : 아키텍쳐 모델들 사이의 상호 관계, 아키텍쳐 모델들과 구현 모델들간의 매핑(abstraction mapping) 등을 유지, 관리한다.
- 일관성 검사기(Consistency Checker) : 저장된 아키텍쳐 모델들 사이에 만족해야 하는 제약조건들의 위반 여부를 검사하고 데이터베이스의 무결성(integrity)를 유지, 보장한다.
- 델타 병합기(Delta Consolidator) : 아키텍쳐 모델들에 대한 유지보수, 정제, 진화 과정 중에 발생한 갱신(updates) 및 추가(increments) 데이터를 델타(delta)들로 유지, 관리하고 외부 요청 시 이들을 베이스 데이터베이스(base database)에 병합한다.

5.4 툴 조정 및 통합 계층(Tool Coordination & Integration Layers)

툴 조정 및 통합 계층은 ADL 지원도구의 다른 계층들에 속한 툴셋과 ADL 지원도구에 의해 사용되는 외부 툴셋 사이의 통신과 상호 작용을 지원하는 시스템 하부구조를 제공한다. 메시지 교환과 이벤트 발생에 의거한 툴 컴포넌트들 사이의 통신을 지원한다.

툴 조정 및 통합 계층의 각 구성요소와 기능은 다음과 같다.

- 툴 등록관리기(Tool Registry) : ADL 지원도구에서 사용될 툴 컴포넌트들을 등록, 관리한다.
- 이벤트 등록관리기(Event Registry) : 이벤트 유형과 각 이벤트 유형에 등록되는 컴포넌트들을 관리한다.



(그림 7) 툴 조정 및 통합 계층

- 이벤트 디스패처(Event Dispatcher) : 이벤트의 발생 시 이를 처리할 컴포넌트들을 이벤트 등록관리기에서 찾아 해당 컴포넌트들의 실행을 트리거(trigger)한다.
- 메시지 전송기(Message Transmitter) : 컴포넌트들 사이의 메시지 송수신을 담당한다.
- 메시지 변환기(Message Translator) : 상호작용이 필요한 컴포넌트들 사이의 메시지 포맷이나 프로토콜이 맞지 않는 경우, 툴 등록관리기의 도움을 얻어 전송할 메시지를 변환(translation)한다.

6. C2 스타일 기반의 ADL 정의와 지원도구의 설계 및 구현

본 장에서는 본 연구팀의 ADL과 5장에서 설명한 ADL 시스템 아키텍쳐의 일부를 구현한 C2 스타일 기반의 ADL 지원도구의 개발 사례를 소개한다.

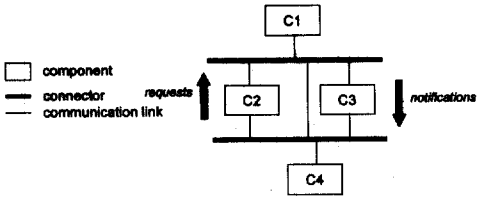
본 연구팀은 먼저, C2 스타일[4]의 아키텍쳐 기술을 지원하는 UCI(University of California in Irvine)의 C2SADL [18]을 자바와 유사한 문법을 갖도록 변경하여 ADL 문법을 순환적 내림차순 파싱(recursive-descent parsing)이 가능한 LL(1) 문법으로 설계한다. 그리고 본 논문에서 제시한 ADL 지원도구의 핵심 기능을 수용할 수 있는 아키텍쳐를 설계하고, 1차적으로 꼭 필요한 부분들을 프로토타입으로 구현한다. ADL 지원도구는 (그림 12)와 같은 구조를 갖으며 ETRI에서 개발 중인 다른 도구들과 통합 가능하도록 UML을 이용하여 설계한다.

6.1 C2 스타일 기반의 ADL

C2 스타일은 UCI에서 본래 GUI를 갖는 응용 소프트웨어 개발을 지원하기 위하여 설계한 아키텍쳐 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 top 포트 또는 bottom 포트를 통해서만 커넥터의 bottom 포트 또는 top 포트에 연결될 수 있다. 따라서 C2 스타일에서는 컴포넌트간의 직접 통신은 허용되지 않는다. 그리고 커넥터는 복수 개의 컴포넌트들 또는 또 다른 커넥터들과 연결될 수 있다.

본 연구팀은 C2SADL과 같이 C2 스타일 기반의 ADL을 1) 컴포넌트 인터페이스 명세 언어(IDN : Interface Definition Notation)와 2) IDN으로 정의된 컴포넌트들과 C2 커넥터들 사이의 바인딩들로 구성된 아키텍쳐 형세(topology)를



(그림 8) C2 아키텍쳐 스타일

기술하는 아키텍쳐 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.

6.1.1 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.

```
<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 9) ADL의 컴포넌트 명세 언어의 구문

(그림 9)는 컴포넌트 명세 언어 즉, IDN의 구문(syntax)이다. (그림 9)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. 컴포넌트의 동작 시작, 또는 종료 시의 행위는 선택적(optional)이고, 포트를 통해 들어오는 메시지에 의한 행위가 아니다. 반면에, 컴포넌트의 실행 시에 행해지는 행위는 컴포넌트의 특정 포트에 들어오는 메시지에 의해 정의된 메소드(들)을 호출하거나 정의된 포트를 통해 메시지(들)을 내보낸다.

6.1.2 아키텍쳐 명세 언어 : ADN(Architecture Definition Notation)

아키텍쳐 명세 언어 즉, ADN은 아키텍쳐의 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 10) ADL의 아키텍쳐 명세 언어의 구문

(그림 10)는 아키텍쳐 명세 언어-ADN의 구문이다. ADN 구문의 컴포넌트 선언 부분은 아키텍쳐 명세에서 사용될 컴포넌트 인스턴스(들)을 열거한다. 각 컴포넌트 인스턴스는 아키텍쳐 상의 위치(top_most, internal, 또는 bottom_most), 컴포넌트 타입, 그리고 컴포넌트 인스턴스명을 갖는다. 만약 컴포넌트 인스턴스의 아키텍쳐 상의 위치가 생략되었다면 해당 컴포넌트 인스턴스의 아키텍쳐 상의 위치는 internal로 간주된다. ADN 구문의 커넥터 선언 부분은 아키텍쳐 명세에서 사용될 커넥터 인스턴스(들)을 열거한다. 각 커넥터 인스턴스는 커넥터 타입, 커넥터 인스턴스명을 갖는다.

ADN의 형세(topology) 부분은 아키텍쳐 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 아키텍쳐 명세의 컴포넌트 또는 커넥터 인스턴스들에 대한 주석(annotation)들로 구성된다.

6.1.3 C2 스타일 기반의 ADL의 특징

본 연구팀은 C2 스타일의 아키텍쳐 기술을 지원하는 UCI의 C2SADL[18]을 변경하여 자바와 유사한 문법 구조를 갖도록 재정의하였다. 본 연구팀의 ADL은 C2SADL에서 제공하는 IDN 및 ADN과 의미적으로 동등한 수준의 C2 스타일 아키텍쳐의 기술을 지원하면서 자바에 친숙한 설계자가 쉽게 사용할 수 있는 구문으로 설계되어 있다. 그러나 현재, 본 연구에서 정의된 ADL에서는 C2SADL에서 아키텍쳐의 동적 변화를 표현하기 위해 제공하는 ACN은 지원하

지 않는다. 반면에, 자바와 동일한 의미의 package문과 import문을 제공한다.

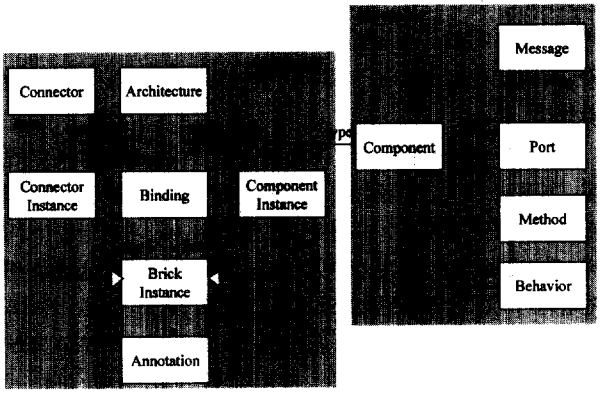
자바의 package문을 도입하여 아키텍처 모델 요소들을 패키지 단위로 그루핑(grouping)하고, import문을 도입하여 다른 패키지에 있는 컴포넌트 명세나 데이터 타입을 import할 수 있게 한다. 따라서, 컴포넌트 자체 개발 뿐만 아니라 컴포넌트 합성을 통한 보다 큰 규모의 시스템 개발이 가능하게 되어, EJB 컴포넌트 기반 시스템 개발의 확장성(scalability)이 향상된다.

본 연구팀의 ADL은 ETRI에서 개발하는 EJB 기반의 컴포넌트 조립 및 생성 지원 도구인 "Cobalt Assembler"의 아키텍처 기술 언어로 사용되고 있다.

6.2 C2 스타일 기반의 ADL 지원도구의 설계 및 구현

본 연구팀이 개발 중인 ADL 지원도구는 본 연구팀의 ADL 처리를 지원하기 위한 도구로서 5장에서 제안한 ADL 지원도구의 시스템 아키텍처 구성 요소 중 뷰/모델 조작 계층(View/Model Manipulation Layer), 그리고 모델 데이터 저장소 계층(Repository Layer)들의 일부 기능을 제공한다. 본 ADL 지원도구는 ETRI에서 개발하고 있는 컴포넌트 조립 도구에서 아키텍처 모델러의 ADL 처리를 지원할 목적으로 개발되었다.

본 장에서는 5장에서 설명한 ADL 시스템 아키텍처의 일부를 구현한 C2 스타일 기반의 ADL 지원도구의 개발 사례를 소개한다.



(그림 11) C2 스타일의 ADL로 기술된 아키텍처의 메타모델

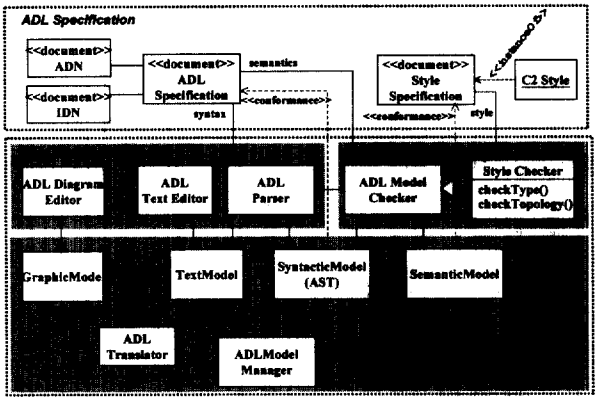
(그림 11)은 본 연구팀의 ADL로 표현되는 아키텍처의 메타모델이다. (그림 1)과 달리 본 연구팀의 ADL은 커넥터를 별도의 모델로, 서브 아키텍처를 갖는 합성 컴포넌트를, 그리고 컴포넌트의 상태 종속적인 행위를 아직 표현하지 못한다. (그림 11)의 모델은 (그림 1)의 아키텍처 메타모델을 C2 스타일에 맞게 특화한 모델로 볼 수 있다.

6.2.1 ADL 시스템의 전체 구조

본 C2 스타일 기반의 ADL 시스템은 (그림 12)과 같이 크게 ADL 명세(ADL Specification)와 이를 지원하는 툴셋인 뷰 조작(View Manipulation) 계층, 모델 조작(Model Manipulation) 계층, 그리고 모델 데이터 저장소(Repository) 계층으로 구성된다.

ADL 명세는 컴포넌트와 아키텍처를 분리하여 기술할 수 있도록 IDN과 ADN으로 정의한 ADL의 구문과 의미에 대한 명세이고, 스타일 명세는 아키텍처에 대한 스타일 명세로서, 본 연구에서는 C2 스타일 아키텍처의 타입 규칙과 형세 규칙에 대한 명세이다.

모델 조작 계층은 그래픽 또는 텍스트 형태의 ADL로 기술된 아키텍처 명세를 편집, 구문 분석하는 도구들로 이루어져 있다.



(그림 12) ADL 시스템의 전체 구조

모델 조작 계층은 ADL 모델 명세에 대한 모델 검사를 지원하는 도구들로 구성되며, 현재 C2 스타일의 타입 오류와 형세 오류를 검사하는 스타일 검사기가 개발 중에 있다.

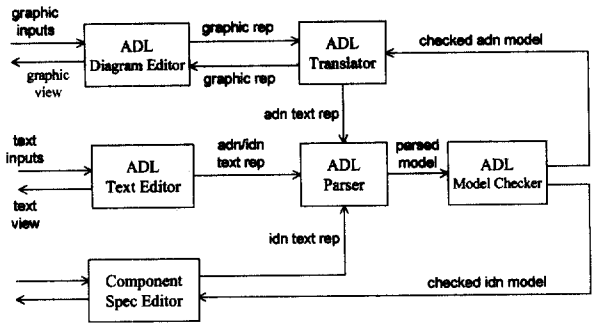
모델 데이터 저장소 계층은 ADL로 기술된 아키텍처 명세의 텍스트 모델, 그래픽 모델, 구문(syntactic) 모델 그리고 시맨틱(semantic) 모델들을 관리하기 위한 도구와 아키텍처 명세에 대한 텍스트 모델과 그래픽 모델들 간의 상호 변환 작업을 지원하는 도구로 구성되어 있다.

6.2.2 ADL 지원도구의 구성 요소

본 연구팀이 개발 중인 ADL 지원도구의 구성 요소들은 다음과 같다.

1) ADL 편집기 : 아키텍처 설계자가 소프트웨어 시스템의 컴포넌트 합성 구조를 ADL 텍스트 형태로 표현하는 작업을 지원하는 ADL 텍스트 편집기와 그래픽 형태로 표현하는 작업을 지원하는 ADL 다이어그램 편집기로 구성된다.
2) ADL 변환기(translator) : 아키텍처 명세의 텍스트 표현과 그래픽 표현 간의 변환과 동기화를 지원한다.
3) ADL 모델검사기 : 작성된 ADL 모델의 구문 오류를 검사하고 파스트리를 생성하는 ADL 파서와, 작성된

ADL 모델의 아키텍쳐 스타일 준수 여부를 검사하는 ADL 스타일 검사기로 구성된다.



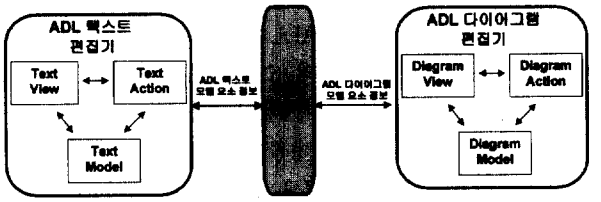
(그림 13) ADL 지원도구의 데이터 흐름

(그림 13)은 ADL 지원도구의 데이터 흐름을 나타낸다. ADL 다이어그램 편집기를 통해 편집된 아키텍쳐 다이어그램은 ADL 변환기의 지원을 받아 텍스트 형태의 ADN 모델 명세로 변환된 후 구문 및 스타일 검사가 수행된다.

(그림 13)에서 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 지원도구의 ADL 파서와 ADL 스타일 검사기를 통해 이루어진다.

6.2.3 ADL 편집기

ADL 편집기는 텍스트 모델의 편집을 위한 ADL 텍스트 편집기와 그래픽 모델의 편집을 위한 ADL 다이어그램 편집기로 구성된다.



(그림 14) ADL 텍스트/다이어그램 편집기와 ADL 변환기

본 ADL 편집기는 MVC (Model-View-Controller) 모델[19]에 의거한 MDI(Multiple Document Interface)를 지원하는 편집 기능을 제공한다.

6.2.4 ADL 변환기

ADL 변환기는 (그림 14)에서 볼 수 있듯이 텍스트 형식의 ADL로 기술된 아키텍쳐 모델과 다이어그램 형태의 ADL로 기술된 아키텍쳐 모델 간의 상호 변환을 지원한다.

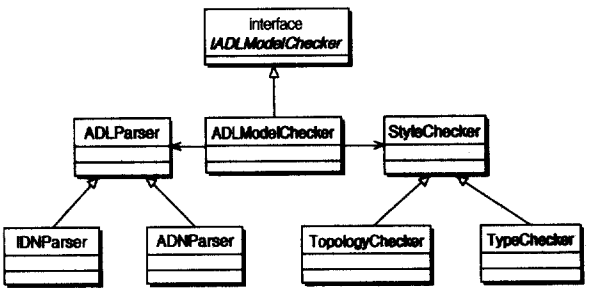
ADL 변환기는 ADL 다이어그램 편집기를 통해 편집된 다이어그램 형태의 아키텍쳐 모델의 요소 정보를 ADL 다이어그램 모델 관리자로부터 받아 ADL 텍스트 모델 관리자에게 전달하거나, ADL 텍스트 편집기를 통해 편집된 텍스트 형태의 아키텍쳐 모델 요소 정보를 ADL 텍스트 모델 관리자로부터 받아 ADL 다이어그램 모델 관리자에게 전달함으로써 ADL 모델의 상호변환을 지원한다.

ADL 모델 간의 실제적인 변환은 ADL 변환기로부터 얻은 ADL 모델 요소 정보를 사용하여 ADL 텍스트 모델 관리자와 ADL 다이어그램 모델 관리자에 의해 이루어진다.

6.2.5 ADL 모델검사기

ADL 모델검사기는 ADL 모델에 대한 구문 검사를 위한 ADL 파서와 C2 아키텍쳐 스타일 명세(타입 및 형세 규칙)의 준수 여부를 검사하기 위한 ADL 스타일 검사기로 이루어져 있다.

(그림 15)는 ADL 모델검사기를 구성하는 인터페이스와 클래스를 표현한 UML 클래스 다이어그램이다.



(그림 15) ADL 모델검사기의 클래스 다이어그램

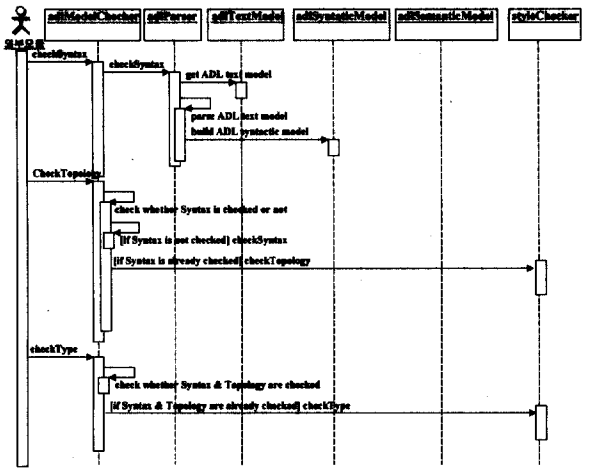
IADLModelChecker는 ADL 검사기에서 외부 모듈에게 제공하는 인터페이스이다. 외부 모듈은 IADLModelChecker 인터페이스만을 통해 ADL 검사기가 제공하는 모델 검사 기능을 이용할 수 있다.

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

(그림 16)은 외부 모듈이 ADL 모델 검사기를 이용하여 구문, 형세, 및 타입 검사를 수행하는 과정을 대략적으로 보여주는 순서도이다.

외부 모듈로부터 IADLModelChecker를 통해 구문 검사 요청이 들어오면 ADL 파서는 ADL 텍스트 모델을 입력 받아 구문 분석하여 ADL 구문 모델(syntactic model)을 생성한다. ADL 텍스트 모델의 구문 검사는 형세 검사의 선행 검사이고, 형세 검사는 타입 검사의 선행 검사이다. 외부 모듈로부터 형세 검사 요청이 들어오면 ADLModelChecker 객체는 구문 검사가 선행되었는지를 점검하여 만약 구문 검사가 선행되지 않았다면 구문 검사를 우선 수행한 후 형세 검사를 수행한다. 타입 검사도 형세 검사와 동일한 방법으로 형세 검사가 선행되었는지를 우선 점검한다.

형세 및 타입 검사는 성공적인 구문 검사 결과로 생성된 ADL 구문 모델을 이용하여 형세와 타입을 검사한다.



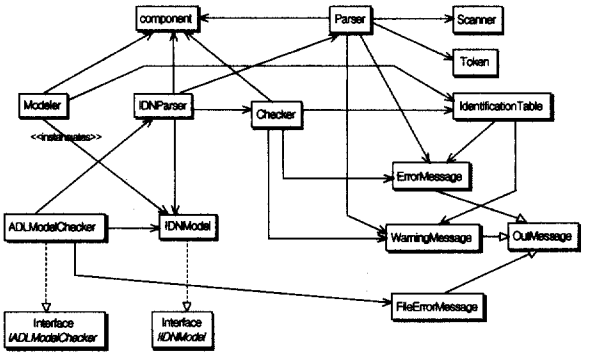
(그림 16) ADL 모델에 대한 구문, 형세 및 타입 검사 과정

(1) ADL 파서

ADL 파서는 컴포넌트 명세를 위한 IDN 파서와 아키텍처 명세를 위한 ADN 파서로 구성되어 있다.

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

(그림 17)은 IDN 파서의 클래스 다이어그램이다.



(그림 17) IDN 파서의 클래스 다이어그램

IDNParser의 구문 검사 기능은 Parser 클래스의 해당 메소드에 구현된다. Parser 클래스는 Scanner 클래스로부터 토큰(token)화 된 IDN 모델 명세를 구문 분석을 하여 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 컨텍스트 분석을 마친 파스트리를 나타내는 IDNModel 객체를 생성한다. 생성된 IDNModel 객체는 타입 및 형세 검사를 위하여 스타일 검사기에 의해 시맨틱(semantic) 모델로 변환된다.

아키텍처 명세에 대한 구문 검사를 수행하는 ADN 파서도 IDN 파서와 유사한 구조를 갖는다.

(2) ADL 스타일 검사기

ADL 스타일 검사는 아키텍처 명세에 대한 형세 및 타입 검사이다.

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

ADL 스타일 검사기는 ADL로 기술된 아키텍처 명세의 시맨틱 모델을 생성한 후, 시맨틱 모델 요소들을 일정한 규칙에 따라 방문하면서 형세 오류와 타입 오류를 검사하는 Visitor 패턴[20]으로 설계되어 있다. 스타일 검사에 적용할 타입 규칙과 형세 규칙들도 검사 코드와 별도로 구성되어 있다. 이러한 설계 방식은 모델 검사기를 구성하는 모델 검사 코드, 스타일 규칙, 그리고 시맨틱 모델이 서로 분리된 구조를 갖게 함으로써, 향후 모델 검사 기능의 변경이나 추가를 용이하게 한다.

7. 결론 및 향후 연구

본 논문에서는 아키텍처 기반의 CBD에 효과적으로 사용될 수 있는 도메인 아키텍처 모델링을 위한 ADL 지원도구의 시스템 아키텍처를 제안하였다. 그리고 C2 스타일의 아키텍처 기술을 지원하는 UCI(University of California in Irvine)의 C2SADL을 변경하여 재정의한 ADL과 지원도구 개발 방법에 대하여 기술하였다. 본 연구팀이 개발 중인 ADL 지원도구는 본 논문에서 제안한 ADL 지원도구의 시스템 아키텍처의 일부 기능을 구현한 개발 사례이다.

본 연구팀의 ADL은 C2SADL이 제공하는 IDN 및 ADN과 의미적으로 동등한 수준의 C2 스타일 아키텍처 기술을 지원하지만, C2SADL에서 아키텍처의 동적 변화를 표현하기 위해 제공하는 ACN은 지원하지 않는다. 그리고 자바(Java)에 친숙한 설계자가 쉽게 사용할 수 있는 ADL을 자바(Java)와 유사한 문법으로 설계하였다. 그리고 자바의 package문을 도입하여 아키텍처 모델 요소들을 패키지 단위로 그루핑(grouping)하고, import문을 도입하여 다른 패키지에 있는 컴포넌트 명세나 데이터 타입을 import할 수 있게 한다. 이를 통하여 컴포넌트 자체 개발 뿐만 아니라 컴포넌트 합성을 통한 보다 큰 규모의 시스템 개발이 가능하게 되어, EJB 컴포넌트 기반 시스템 개발의 확장성(scalability)이 향상된다.

본 연구팀의 ADL 지원도구는 CBD에 아키텍처를 접목하여 개발하였다. 따라서, 본 연구팀의 ADL 지원도구는 아키텍처 기반의 CBD를 효과적으로 지원하고, 아키텍처를 정확하고 엄밀하게 모델링, 분석, 검증할 수 있는 능력을 향상시키고, 독자적인 ADL 지원환경을 구축할 수 있는 기반을

제공한다. 또한 본 연구팀의 ADL과 지원도구는 컴포넌트 개발 및 조립 지원도구의 도메인 아키텍쳐 기술 언어로 활용하여 텍스트 또는 그래픽 표기 형식으로 기술된 컴포넌트 명세와 아키텍쳐 명세에 대한 모델 검사기로 사용될 수 있다.

본 연구팀의 ADL과 지원도구는 현재, ETRI에서 개발하는 EJB 기반의 컴포넌트 조립 및 생성 지원 도구인 "Cobalt Assembler"의 아키텍쳐 기술 언어와 지원도구로 사용되고 있다. 현재 ETRI의 Cobalt Assembler에 잘 통합되어 시험 중에 있다. ETRI는 현재, 본 연구팀의 ADL 지원도구가 통합된 Cobalt Assembler를 시험적으로 이용하여 쇼핑몰을 성공적으로 개발하였다.

향후에는, 본 연구팀의 ADL를 아래에 열거한 표현 능력을 가질 수 있게 확장, 일반화하여 소프트웨어를 아키텍쳐 수준에서 설계, 분석, 정제, 진화하는데 효과적으로 사용할 수 있는 ADL로 발전시킬 계획이다.

- 아키텍쳐 명세에서 커넥터의 의미를 컴포넌트와 동등한 수준에서 직접적으로 표현할 수 있는 능력.
- 서브 아키텍쳐를 갖는 합성 컴포넌트(composite component)를 표현할 수 있는 능력.
- 아키텍쳐 수준에서 컴포넌트와 시스템 행위의 의미를 정확히 표현할 수 있는 능력.
- 아키텍쳐의 동적 변화를 표현할 수 있는 능력.
- C2가 아닌 스타일의 아키텍쳐도 기술할 수 있는 능력.

그리고 본 연구팀의 C2 스타일 기반의 ADL 지원도구는 ETRI에서 개발하고 있는 컴포넌트 조립 지원도구에서 아키텍쳐 모델러의 ADL 처리기로 사용할 목적으로 개발되었다. 6장에서 설명한 본 ADL 지원도구의 구성 요소들 중 스타일 검사기를 제외한 나머지 부분들이 현재 프로토타입으로 구현되어 있다. 그리고 컴포넌트 조립 지원도구의 나머지 부분들과 통합 시험 중에 있다. ADL 스타일 검사기는 개발 중에 있다.

향후, 본 논문에서 기술한 ADL 지원도구를 본 논문에서 제안한 ADL 시스템 아키텍쳐의 나머지 부분들도 지원하는 ADL 지원도구로 확장, 개발하기 위한 연구를 진행할 계획이다.

참 고 문 헌

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

[2] Nenad Medvidovic and Richard N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE Transactions on Software Engineering, Vol.26, No.1, pp.70-93, January, 2000.

[3] Oh-Cheon Kwon, Gyu-Sang Shin, and Taewoong Jeon, "An Open Architectural Framework for Building an Architecture Modeler," Proceedings of the 15th International Symposium on Computer and Information Sciences(ISCIS 2000), October 11-13, 2000, Turkey Istanbul, Yildiz Technic University, pp.477-486, "Description Languages," IEEE Transactions on Software Engineering, Vol.26, No.1, pp. 70-93, January, 2000.

[4] 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, pp.390-406, June, 1996.

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

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

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

[8] R. Allen and D. Garlan, "Beyond Definition/Use : Architectural Interconnection," Proceedings of Workshop on Interface Definition Languages, Portland, Oregon, USA, January, 1994.

[9] 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.

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

[11] 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, pp.314-335, April, 1995.

[12] 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.

[13] 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, pp.390-406, June, 1996.

[14] B. Selic, G. Gullekson, and P. T. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, Inc., 1994.

[15] S. Vestal, MetaH Programmer's Manual, Version 1.09,

Technical Report, Honeywell Technology Center, April, 1996.

[16] J. E. Robbins, N. Medvidovic, D. F. Redmiles, and D. S. Rosenblum, "Integrating Architecture Description Languages with a Standard Design Method," Proceedings of the 20th International Conference on Software Engineering, Kyoto, Japan, pp.209-218, April, 1998.

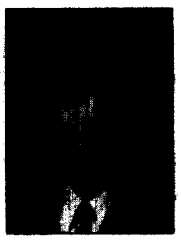
[17] D. Garlan, R. Monroe, and D. Wile, "Acme : An Architecture Description Interchange Language," Proceedings of CASCON'97, pp.169-183, Nov. 1997.

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

[19] G. Krasner and S. Pope. A Cookbook for Using the MVC User Interface Paradigm in Smalltalk. Journal of Object-Oriented Programming, 1(3) : 26-49, 1988.

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

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



신 동 익

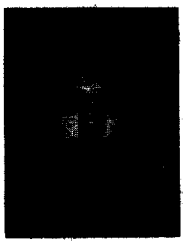
e-mail : eastwing@selab.korea.ac.kr

1996년 고려대학교 전산학과(학사)

1998년 고려대학교 전산학과(석사)

1998년~현재 고려대학교 전산학과 박사 과정

관심분야 : 소프트웨어 시험, 소프트웨어 아키텍처, 객체지향 프레임웍, 정형 명세 등



노 성 환

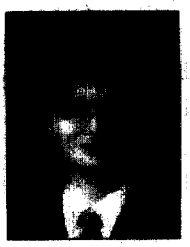
e-mail : shroh@selab.korea.ac.kr

1997년 고려대학교 전산학과(학사)

1999년 고려대학교 전산학과(석사)

2000년~현재 고려대학교 전산학과 박사 과정

관심분야 : 소프트웨어 시험, 소프트웨어 아키텍처, 객체지향 프레임웍, 설계 패턴 등



최 재 각

e-mail : choi@selab.korea.ac.kr

2000년 고려대학교 전산학과(학사)

2000년~현재 고려대학교 전산학과 석사 과정

관심분야 : 객체지향 소프트웨어공학, 소프트웨어 아키텍처 등

전 태 웅

e-mail : jeon@tiger.korea.ac.kr

1981년 서울대학교 계산통계학과 (학사)

1983년 서울대학교 계산통계학과 (석사)

1992년 Illinois Institute of Technology, Dept. Computer Science, Ph. D

1983년~1987년 금성통신(현 LG전자) 연구소 주임연구원

1992년~1995년 LG산전 연구소 책임연구원

1995년~현재 고려대학교 전산학과 부교수

관심분야 : 소프트웨어 시험, 객체지향 프레임웍, 소프트웨어 아키텍처, 실시간 소프트웨어공학, 공정제어 시스템 등.

이 승 연

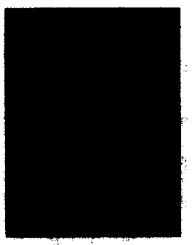
e-mail : coral@etri.re.kr

1999년 서강대학교 전자계산학과 공학사

2001년 서강대학교 컴퓨터학과 공학석사

2001년~현재 한국전자통신연구원 컴퓨터·소프트웨어기술연구소 S/W 공학 연구부 연구원

관심분야 : 소프트웨어 아키텍처, 소프트웨어 개발방법론, 컴포넌트 응용, Web Services 기술 등



권 오 천

e-mail : ockwon@etri.re.kr

1994년 영국 Teesside 대학교 S/W공학과 공학석사

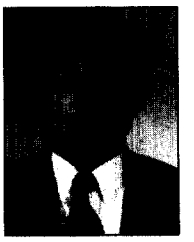
1998년 영국 Durham 대학교 전산과학과 공학박사

1991년 미국 RTP IBM 연구소 객원 연구원

1993년 시스템공학연구소 선임연구원

현재 ETRI 컴퓨터·소프트웨어기술연구소 S/W공학연구부 책임연구원

관심분야 : 재사용, CBD, Product Line, Web Services 기술 등



신 규 상

e-mail : gsshin@etri.re.kr

1981년 성균관대학교 통계학과 학사

1983년 서울대학교 계산통계학과 이학석사

2001년 충남대학교 컴퓨터과학과 이학박사

1983년 시스템공학연구소 연구원

1987년 시스템공학연구소 선임연구원

1997년 한국전자통신연구원 책임연구원

현재 ETRI 컴퓨터·소프트웨어기술연구소 S/W 공학연구부 컴포넌트공학연구팀 팀장

관심분야 : CASE, S/W 컴포넌트 기술, 웹 서비스 기술, 멀티미디어 시스템 등