ADL을 이용한 프레임워크의 소프트웨어 아키텍처 표현에 관한 연구

김태균°, 김강태, 이경환
중앙대학교 컴퓨터공학과 소프트웨어공학 연구실

A Study on the Software Architecture Representation of Framework Using the ADL

TaeGyun Kim, KangTae Kim, KyungWhan Lee
SE Laboratory, Dept of Computer Science & Engineering, Chung-Ang University

요 약

프레임워크는 객체 지향 재사용 방법 중의 하나로서 사용되고 있다. 프레임워크는 소스 코드의 재사용뿐만 아니라 지식의 재사용이라는 측면에서 보다 높은 수준의 재사용이다. 여기서 소스 코드의 재사용이 객체 지향 중심에서 컴포넌트 기반으로 바뀌어 가는 추세에 맞추어 프레임워크 역시 객체 지향 프레임워크에서 컴포넌트 프레임워크로 바뀌어 가고 있다. 컴포넌트 프레임워크는 객체 지향 프레임워크와는 달리 컴포넌트들의 컴포지션에 의한 어플리케이션 개발로서, 컴포넌트간의 연결성을 명세함으로써 프레임워크의 아키텍처를 보다 상세하게 표현할 필요가 생겼다.

본 논문은 이러한 컴포넌트 컴포지션에 필요한 프레임워크의 아키텍처 정보를 표현하는 한가지 방법으로서 ADL을 통해 프레임워크의 아키텍처를 표현하고, 이 ADL을 다른 프레임워크 보조 도구와 함께 사용하면서 컴포넌트 컴포지션을 보다 용이하게 함으로써, 어플리케이션을 개발하는 것과 보다 높은 수준의 재사용을 가능하게 하고자 한다

1. 서론

객체 지향 재사용의 한가지 방법으로서 프레임워크를 기반으로 한 어플리케이션의 개발은 단순한 코드 수준의 재사용뿐만 아니라 영역에 대한 지식을 재사용 하는 보다 높은 수준의 재사용을 지원한다. 프레임워크는 객체 지향 기술 중 상속 메커니즘과 동적 바인딩, 그리고 디자인 패턴을 이용하여 어플리케이션의 설계 지식을 재사용 가능하게끔 하였다. 일반적으로 프레임워크라 함은 객체 지향 프레임워크를 의미한다. 하지만 이런 객체 지향 프레임워크와 컴포넌트 프레임워크에는 엄연한 차이점이 있다[1]. 특정 문제에 대한 해결책으로서 디자인 패턴을 이용하여 재사용의 기능성에 초점을 맞춘 객체 지향 프레임워크와는 달리 컴포넌트 프레임워크는 컴포넌트의 확장에 대한 규칙을 정의함으로서 추상 개념을 구체화하는 데 주안점을 두고 있다

그리고 기존의 객체 지향 프레임워크를 사용할 경우 'hot spot'에 대해서는 프레임워크 사용자가 서브클래싱 메커니즘을 이용하여 적용할 수 있는 인터페이스 클래스의 집합을 제공한다. 이와 같은 방식은 기존의 객체 라이브러리를 이미 조직 내에서 가지고 있을 경우 이 라이브러리 객체들을 다른 클래스로 랩핑하여 사용하거나 추가적인 중간 객체를 만들어 사용하는 방법 밖에 없다. 이런 접근 방법에서 일반적으로 상속 메커니즘을 사용하여 프레임워크의 인터페이스 클래스의 인터페이스를 상속한다. 어떤 방법을 사용하건 간에 라이브러리 클래스나, 컴포넌트를 인터페이스 클래스에서 상속되는 것으로 만들 수는 없다. 또한 프레임워크의 해당 어플리케이션을 사용자가 원하는 데로 개조할 수도 없게 된다[2].

그리고 디자인 패턴은 특정 문제에 대한 해결책으로서, 그리고 프레임워크 개발의 디자인을 결정하는 데 있어 디자인 정보 교환의 한 방법으로서 사용되어 왔다. 또한 이런 디자인 패턴은 문제점에 대한 해결책으로서 아키텍처를 유도해 냄으로서 사용되어 왔다[3]. 그러나 이와 같은 디자인 패턴이 문제 해결에 대한 아키텍처와 해결책을 제안하여 주기는 하지만, 전체적인 프레임워크의 아키텍처를 이해하고, 또한 부분적인 아키텍처간의 연결성을 이해하는 데는 오히려 디자인 패턴의 개념과 실제 아키텍처 개념과의 혼동으로 구현물에 대한 추적성에 역영향을 끼치기도 한다[4]

기존의 객체 지향 설계 방법에서는 컴포넌트간의 상호작용에 대한 정보를 특색화 하기 힘들다. 그리고 코드에 설계자의 모든 개념을 반영할 수 없고, 단지 설계자가 이해한 시스템 호출이나 다른 하위 수준의 메커니즘에 따르는 수밖에 없다. 연결에 대한 정보가 이런 방법에 의해 흩어지면, 시스템 연결성

을 구분하기 힘들어 지고, 연결 메커니즘은 재사용되기 힘들다. 그리고 컴포넌트가 내재된 연결 정보를 가지게 된다면 재사용되기 힘들어 진다[5].

이에 대한 해결책으로 본 논문에서는 먼저 객체 지향 프레임워크에서 컴포넌트 기반의 프레임워크로의 진환을 위해 각각의 차이점을 비교하고, 프레임워크의 지식이라 할 수 있는 아키텍처에 대한 정보를 표현하는 방법 중의 하나로서 ADL을 이용하여 프레임워크 아키텍처를 표현하였다. 그리고 이렇게 표현된 ADL을 프레임워크의 프로그래밍 언어와 CASE 도구와 연관하여 사용할 수 있도록 하였다.

2. 컴포넌트 기반 프레임워크

프레임워크는 객체 지향 재사용 기술이라고 할 수 있다. 그러나 프레임워크는 기존의 객체 지향 재사용 기술과는 다른 점을 가지고 있다. 이상적인 재사용 기술은 컴포넌트를 사용하여 새로운 시스템을 구축할 때 컴포넌트간의 연결을 용이하게 할 수 있도록 하는 데 있다. 소프트웨어 개발자는 컴포넌트가 내부적으로 어떻게 구현되었는가는 알 필요 없이 외부적인 사용 방법만 쉽게 익혀서 사용하는 것이다. 결과적으로 컴포넌트를 이용하여 구축한 시스템은 보다 효율적이 되고, 유지 보수하기 쉬울 뿐만 아니라, 안정적이 된다.[6]

이러한 장점 때문에 프레임워크 역시 객체 지향에서 컴포넌트 지향으로 그 추세가 바뀌고 있다. 엄밀히 말해서 객체 지향 프레임워크와 컴포넌트 프레임워크는 각기 다른 점이 존재한다. 객체 지향 프레임워크는 객체 지향 언어에서의, 문제에 대한 해결책으로 사용되는 디자인 패턴에 대한 구현이다 즉 재사용을 제공하기 위한 기능으로서 초점이 맞추어져 있다. 그러나 컴포넌트 프레임워크는 컴포넌트들을 확장하는 규칙을 정의한 추상 개념을 구체화하는 데 초점이 맞추어져 있다.

그러나 프레임워크는 특성상 커스터마이징해야 할 부분이 많다는 어려움이 있다. 이는 보다 다양한 컴포넌트들을 확장하기 쉽게 설계되어야 한다는 것이다. 다시 말해서 컴포넌트 프레임워크는 설계 시부터 컴포넌트들의 제약 조건, 연결성 등과 같은 규칙들을 명세하여야 한다.

3. ADL(Architecture Description Language)의 적용

아키텍처란 컴포넌트와 이 컴포넌트들간의 상호 작용을 정의한 것이다. 어떠한 소프트웨어도 각각 아키텍처를 가지고 있고 프레임워크 역시 마찬가지이다. 컴포넌트 프레임워크의 경우 특히 컴포넌트의 확장에 의한 어플리케이션 개발이 이루어지기 때문에 이러한 아키텍처에 대한 정보가 기술되어 사용되면 보다 용이한 개발이 가능하다

기존의 프레임워크에서도 이러한 아키텍처를 디자인 패턴과 같은 방법을 사용하여 나름대로 정의하여 아키텍처를 표현함으로써 보다 높은 수준의 재사용을 이끌어 온 것 사실이다. 그러나 디자인 패턴만을 이용한 아키텍처 표현에는 한계가 있다. 아키텍처들 간의 연결성과 시스템의 전체 아키텍처에 대한 정보 등을 표현하기에는 부족하다. 특히 컴포넌트 컴포지션을 이용한 어플리케이션 개발을 하는 컴포지션 기반의 프레임워크에서는 주어진 문제에 대한 해결책으로서 제안된 디자인 패턴만으로는 그 표현에 한계가 있다. 컴포넌트의 상호 작용이 계층화된 아키텍처(layered architecture)로 이루어져 있는 경우에는 UML과 같은 언어로는 표현하는 데 어려움이 있지만 ADL과 같은 언어의 경우 쉽게 표현이 가능하다. 따라서 본 논문에서는 이런 아키텍처의 표현뿐만 아니라 상호 연결성을 표현하기 위해 ADL을 사용하였다.

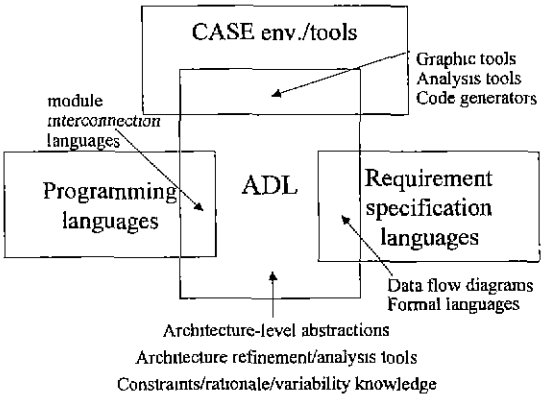

그림 1 ADL과 다른 도구, 언어와의 관계[7]

그림 1에서 볼 수 있듯이 ADL은 단순히 아키텍처만을 표현하는 데 그치지 않고 프로그래밍 언어나, 요구 명세 언어와 서로 연관하여 사용될 수 있고, 또한 CASE 도구와 같이 혼용하여 사용할 수 있다.

실제로 ADL이 컴포넌트 프레임워크에서 컴포넌트의 확장을 위해 다른 CASE 도구나 프로그래밍 언어 등과 같이 연관하여 사용되어지는데 이를 위해서는 다음과 같은 사항이 명세되어야 한다

3.1 컴포넌트(component)

컴포넌트는 기본 컴포넌트(primitive component)와 복합 컴포넌트(composite component)로 분류된다. 복합 컴포넌트는 가장 기본이 되는 기본 컴포넌트들을 컴포지션함으로서 생성된 컴포넌트를 의미한다.[8]

모든 컴포넌트는 사용되는 컴포넌트의 이름과, 상위 객체 인터페이스를 정의한 컴포넌트의 이름, 그리고 컴포넌트가 제공하는 인터페이스와 요구하는 인터페이스와 같은 컨트랙트가 명세되어야 한다

복합 컴포넌트는 여러 컴포넌트들간의 관련성이 정의된다. 이 관련성은 커넥터로서 표현된다

3.2 커넥터(connector)

컴포넌트들간의 관련성을 정의한다. 커넥터 역시 기본 커넥터(primitive connector)와 복합 커넥터(composite connector)로 구분된다.

이와 같은 정보를 표현하는 특성을 만족하는 여러 ADL중에서 본 논문에서는 UniCon(UNIversal CONnector support)을 사용하여 아키텍처를 표현하였다. 그림 2는 채팅 어플리케이션의 복합 컴포넌트를 표현한 것이고, 그림 3은 두 개의 컴포넌트를 연결시키는 커넥터를 표현한 것이다.

```
COMPONENT ChatClient
  INTERFACE IS
  TYPE java.applet.Applet
      PLAYER initContents IS publicFunction

  END INTERFACE
  IMPLEMENTATION IS
      USES inputDlg INTERFACE inputDialog
      USES showDlg INTERFACE showDialog
      USES conn INTERFACE connClient
      USES hook1 INTERFACE HookUp9951

      CONNECT inputDlg TO hook1 source
      CONNECT conn.addConnEventListener TO hook1 intf1
      ...
  END IMPLEMENTATION
END reverse-filter
```

그림 2 UniCon으로 표현한 컴포넌트

```
CONNECTOR HookUp9951
  PROTOCOL IS
  TYPE ChattingEventListener
      ROLE source IS setTarget
      ROLE intf1 IS inputDialog
      .
  END PROTOCOL
  IMPLEMENTATION IS
      BUILTIN
  END IMPLEMENTATION
END HookUp9951
```

그림 3 UniCon으로 표현한 커넥터

이렇게 프레임워크의 아키텍처를 ADL로 표현함으로써, 개발하는 어플리케이션의 컴포넌트 구성과, 컴포넌트간의 관련성을 쉽게 이해할 수 있다. 또한 표현된 ADL을 다른 프로그래밍 언어와 CASE 도구와 같이 연관하여 사용하여 어플리케이션 개발의 용이성을 높였다.

4. ADL과 다른 언어, 도구와의 연계

앞서 언급한 것처럼 ADL은 프레임워크의 아키텍처를 표현함으로서 이해도를 높이는 데에 사용되는 것뿐만 아니라 다른 CASE 도구와 프로그래밍 언어와 함께 사용된다.

컴포넌트와 커넥터에 대한 명세는 각각 UML notation과 연관되어 프레임워크의 가시적인 이해를 위한 도구에 쓰일 수 있다. 그리고 ADL은 프로그래밍 언어와 연관되어 컴포넌트를 컴포지션하는 프로그래밍 언어 생성기의 입력으로도 사용된다. 이때 컴포넌트간의 연결성을 디자인 패턴의 아답터 패턴(adapter pattern)이나 브리지 패턴(bridge pattern) 등과 같은 패턴 개념을 사용하여 프로그래밍 언어로 바꾸어 생성함으로서 컴포넌트 컴포지션을 시켰다.

5. 결론 및 향후 과제

컴포넌트 프레임워크는 컴포넌트의 확장에 의한 어플리케이션 개발에 초점을 맞추어 개발되었다. 이런 컴포넌트의 컴포지션을 지원하기 위해서는 컴포지션을 위해 필요한 모든 정보가 ADL을 이용하여 정형화된 형태로 표현되어야 하고 이 표현을 기반으로 하여 컴포넌트 컴포지션을 하였다. 그리고 이러한 ADL을 통해 아키텍처를 표현하고, 또한 아키텍처 간의 상호 연결성을 표현할 수 있기 때문에 이를 통한 단순한 코드 수준의 재사용 보다 높은 수준의 재사용을 얻어내었다.

프레임워크의 아키텍처를 ADL로서 컴포넌트의 상호 작용 등을 표현함으로써, 프레임워크의 이해성을 높인다는 것과 ADL을 다른 CASE 도구와 연관하여 사용함으로써 컴포넌트 컴포지션을 용이하게 할 수 있다는 장점이 있다.

그러나 이런 아키텍처 정보를 보다 재사용성을 높이기 위해서는 아키텍처 자체를 컴포넌트화하는 작업이 필요하다. 그리고 이를 위해서는 또한 아키텍처들간을 연결시켜주는 커넥터에 대한 연구도 필요하다.

6. 참고문헌

[1] Wolfgang Weck, "Independeltly extensible component frameworks", Workshop on Component-Oriented Programming WCOP' 96

[2] C. Lundberg, M. Mattsson, "Using Legacy Components with Object-oriented Frameworks," Proceedings of Systemarkitekturer '96 Conference, Boras, Sweden, 1996.

[3] Kent Beck, Ralph Johnson, "Patterns generate architectures", ECOOP, 1994

[4] Jan Bosch, "Specifying Frameworks and Design Patterns as Architectural Fragments", Proceedings TOOLS ASIA '98

[5] Mary Shaw, Robert DeLine, Gregory Zelesnik, "Abstractions and Implementations for Architectural Connections", Proc Third International Conference on Configurable Distributed Systems, May 1996.

[6] Ralph E. Johnson, "Frameworks = (Components + Patterns)", Communication of the ACM, 1997

[7] Paul Kogut, Paul Clements, "Features of Architecture Description Language", Software Technology Conference, 1995

[8] M Shaw, R DeLine, D Klein, T.Ross, D Young, and G.Zelesnik, "Abstractions for Software Architecture and Tools to Support Them", IEEE Transactions on Software Engineering, April 1995