소프트웨어 아키텍처를 적용한 컴포넌트 프레임워크 개발에 관한 연구

이 창 훈† · 이 경 환††

요 약

프레임워크는 기본적으로 소프트웨어 개발 시 얻어진 생산물들을 상위레벨의 추상화 과정을 통해 그 분석 및 설계 정보의 재사용을 통해 소프트웨어의 재사용 범위를 넓히고자 하는 시도이다. 그러나 이 프레임워크는 역호출관계를 통한 어플리케이션 개발 시스템이기 때문에 어플리케이션에 대한 아키텍처 정보를 모두 가지고 있어야 한다. 기존의 프레임워크의 경우 이런 아키텍처 정보는 설계 수준에서만 머물고 바로 코드 수준에서 정의, 사용되는 형태를 가져왔다. 따라서 프레임워크의 확장이나 컴포지션 시 코드를 재 설계하고 구현해야 하는 문제점이 있다. 즉 아키텍처 정보를 설계 수준과 코드 재사용 수준의 중간 형태인 언어로서 개발, 사용해야할 필요성이 생겼다. 본 논문에서는 다음과 같은 연구에 중점을 두었다. 첫째로 아키텍처 정보를 보다 구체화하는 방법으로 ADL을 통한 표현 방법에 대해 정의하였다. 둘째로 기존의 추상화된 컴포넌트 기반 프레임워크 개발 공정을 아키텍처 정보를 구체화하여 개발하는데 적합하도록 그 공정을 개선하였다. 셋째로 ADL로 표현된 아키텍처 정보를 프레임워크를 통한 어플리케이션 개발에 필요한 정보로 활용하여 개발할 수 있도록 컴포넌트 프레임워크 지원도구를 개발하였다.

A Study on the Building Component Framework Development adapting Software Architecture

Chang-Hoon Lee† · Kyung-Whan Lee††

ABSTRACT

The reuse by using framework is proposed to improve productivity. It is a set of usable and expandable classes and their connectivity. But frameworks are described with programming languages, it is hard for developers to learn the collaborative patterns of a framework by reading it. Patterns are one approach to improving the documentation. But this should be redesigned to expand and redefine the framework. The necessity of the formal description of architecture information is being proposed to relate to programming language. This paper support the following points. First of all, it has been proposed the description of the needed elements when developing a framework using ADL. Secondly, the current development process has been refined to be suitable for developing the domain framework. Thirdly, it has been proposed the development of a application using a framework implemented by an architecture information described with ADL. Overally, the main contents of this research is defining the ideas of a description and development of an architecture framework using ADL.

키워드 : 컴포넌트(component), 소프트웨어 아키텍처(Software architecture), 프레임워크(Framework), ADL

1. 서 론

객체 지향 재사용의 한 가지 방법으로서 프레임워크를 기반으로 한 어플리케이션의 개발은 단순한 코드 수준의 재사용뿐만 아니라 영역에 대한 지식을 재사용하는 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 구체화하는데 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

† 정 회 원 : 중앙대학교 정보통신연구소 연구전담교수
†† 정 회 원 : 중앙대학교 컴퓨터공학과 교수
논문접수 : 2001년 11월 17일, 심사완료 : 2002년 1월 16일

사용하는 방법밖에 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[2].

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.

한편, 소프트웨어 공학은 표준 개발 프로세스와 표준 부품을 통해서 제품을 생산하는 하드웨어 생성 공정과 같이 특정 프로그램 군에 대한 표준적인 프로세스와 표준 부품을 통해 효율적인 프로그램 개발을 목표로 하고 있다. 이러한 목표를 달성하기 위한 노력 중 하나가 소프트웨어 아키텍처이다. 소프트웨어 아키텍처는 소프트웨어 아키텍처를 구성하는 컴포넌트, 이들간의 관련사항을 제어하는 커넥터, 컴포넌트 및 커넥터가 가지는 제약사항, 이들을 설계하고 구현하는 데 적용되는 아키텍처 스타일과 설계패턴에 의해서 소프트웨어 시스템 관련자의 요구사항을 만족시킬 수 있는 소프트웨어 시스템의 구조를 제공하는 소프트웨어 부품의 집합이며 이러한 소프트웨어 부품을 이용해서 소프트웨어를 개발하는 공정을 의미한다.

이와 같은 맥락에서 본 논문에서는 먼저 객체 지향 프레임워크에서 컴포넌트 기반의 프레임워크로의 전환을 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. 관련 연구

프레임워크는 객체 지향 재사용 기술 중의 한 가지 방법이다. 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 사용방법만 쉽게 익혀서 사용하는 것이다. 결과적으로 컴포넌트를 이용하여 구축한 시스템은 보다 효율적이 되고, 유지·보수하기 쉬울 뿐만 아니라, 안정적이 된다[3].

2.1 프레임워크 개발에 필요한 사항

프레임워크는 객체 지향 시스템의 아키텍처 디자인이다. 이는 시스템의 컴포넌트와 이들간의 상호작용을 나타낸다. 프레임워크에서 클래스는 시스템의 컴포넌트를 정의한다. 시스템의 상호작용은 제약조건(constraints), 상속(inheritance), 그리고 컴포지션(compistion)의 비정형적인 규칙으로 정의된다[4]. 이와 같은 정의 내용은 다음과 같은 크게 세 가지의 사항으로 정리된다.

2.1.1 추상 클래스(Abstract class)

추상 클래스는 구체화된 클래스(concrete class)에 대한 일반화된 인터페이스를 제공한다. 추상 클래스의 프레임워크는 시스템에서의 객체들간의 제약조건을 만들어 낸다.

2.1.2 제약조건(Constraints)

제약조건은 프레임워크의 추상 클래스들간의 관련성이나 프레임워크 내에서의 구체화된 클래스와 추상 클래스간의 관련성을 나타낸다.

2.1.3 동적 코드 로딩(Dynamic code loading)

동적 코드 로딩은 시스템을 디자인할 때, 추상 클래스를 명시하여 이 추상 클래스에 대한 상속된 클래스에서 상속된 구체화된 클래스를 추가할 수 있도록 한다.

2.2 프레임워크 기반 프로그래밍

프레임워크를 기반으로 한 프로그래밍은 프레임에서 변화하지 않는 부분과 변화 가능한 부분을 구분하여 변화 가능한 부분에 대한 변화를 적용하는 메커니즘을 제공한다.

프레임워크를 개발하는 개발 프로세스의 전체적인 단계는 다음과 같다[5].

2.2.1 영역 구분 및 특성화

프레임워크는 문제에 대한 가능한 솔루션의 추상 개념으로, 현재 솔루션을 분석함으로써 필요한 프레임워크를 정하기 위해서 어플리케이션의 집합으로서 개발 영역을 분석하는 과정이다.

2.2.2 아키텍처를 정의하고 설계한다.

분석 과정에서 찾은 객체를 어떻게 논리적으로 정의해야 구현이 가능할지를 결정한다.

2.2.3 프레임워크 구현

구현 단계는 프로그래밍 언어로 논리적 설계를 실제적 해결 대상으로 만드는 것이다.

2.3 소프트웨어 아키텍처

일반적으로 소프트웨어 아키텍처는 소프트웨어 아키텍처를 구성하는 컴포넌트, 이들 간의 관련사항을 제어하는 커넥터, 컴포넌트 및 커넥터가 가지는 제약사항, 이들을 설계하고 구현하는데 적용되는 아키텍처 스타일과 설계 패턴에 의해서 소프트웨어 시스템의 관련자와 요구사항을 만족시킬 수 있는 소프트웨어 시스템의 구조를 제공하는 소프트웨어 부품의 집

합이며 이러한 소프트웨어 부품을 이용해서 소프트웨어를 개발하는 공정을 정의한 것이라고 할 수 있다[9].

따라서 소프트웨어 아키텍처는 개발의도와 사용 방법에 의해서 크게 두 가지 관점을 가지고 있다고 볼 수 있다.

첫째, 소프트웨어 아키텍처를 설계 전략의 관점에서 보는 것과 둘째, 소프트웨어 아키텍처를 추상화 메카니즘의 하나로 보는 관점이다.

한편, 기존 아키텍처 기술(description) 기법으로는 다음과 같은 언어들을 사용한다.

2.3.1 UML

UML(Unified Modeling Language)은 소프트웨어 중심 시스템의 산출물을 가시화하고, 명세화하며, 구축하고, 그리고 문서화하는데 사용되는 표준 언어이다. UML은 표현력이 매우 풍부한 언어로서, 시스템들을 개발하고 배치하는데 필요한 모든 관점을 다룬다[7].

UML을 통해서 소프트웨어 중심 시스템의 아키텍처는 상호 연결되는 "4+1 View"로 설명될 수 있다. 각 뷰는 시스템 조직과 구조를 투영하고 있으며, 그 시스템의 특정한 관점에 초점을 둔다.

2.3.2 MIL/IDL

일반적으로 MIL과 IDL은 잘 정의된 인터페이스를 가지는 컴퓨팅 단위와 조각들을 연결(glue)하는 합성 메커니즘을 나타내는 표기법을 제공하는 역할을 하여왔다. 장점으로는 사용되고 정의된 기능들이 프로그래밍 언어에서 선택될 수 있기 때문에, 현재 프로그래밍 언어와 잘 맞는다는 것이다. 문제점은, 시스템의 각 부분간의 구현 관련성은 잘 기술하는 반면, 아키텍처 표현의 중심이 되는 상호작용 관련성을 기술하는 데는 적합하지 못하다는 것이다[8].

2.3.3 ADL

아키텍처 기술 언어(ADL)는 아키텍처 모델에 대한 표시법으로 생겨나고 있다. ADL은 그래픽적인 표현과 문자적인 표현을 사용하여 아키텍처 정보를 나타낸다. ADL은 그들이 지원하는 아키텍처 스타일과 그들이 제한하는 분석 형태에 따라 다양하게 바뀐다. 다른 도구들처럼, 모든 가능한 상황에 최적으로 맞는 ADL은 없다. 다양한 그룹에서 ADL을 사용한다. ADL은 설계, 개발, re-engineering 어플리케이션 시스템에 대한 중요한 도구로서 사용된다.

현재 많은 기업, 연구소에서 다양한 ADL이 만들어지고 있으며, 현재까지 소개된 ADL로는 Rapide, Unicon, GenVoca, MetaH, Aesop, 그리고 C2등이 있다. 이들 중에서 가장 보편적으로 사용되는 Rapide와 Unicon에 대하여 알아보면 Rapide는 분석, 모델링, 시뮬레이션, 코드 생성을 지원하는 툴셋이 제공되는 것이 특징이며 확정성이 뛰어나다는 장점을 가지고 있으나 소프트웨어 아키텍처의 중요한 요소인 커넥터에 대한 정의가 미약하다는 단점을 가지고 있다. Unicon은 확장성이 떨어지고 특정한 영역에 적용될 수 있다는 단점을 가지고 있으나 커넥터를 하나의 클래스로 정의하고 있고 여러 종류의 미리 만들어진 커넥터를 제공한다는 장점을 가진다[10].

3. 소프트웨어 아키텍처를 적용한 프레임워크 개발 기술

3.1 프레임워크 설계/개발 과정과 ADL

2장의 2.1절에서 기술한 것과 같이 프레임워크를 개발하기 위해서는 크게 추상 클래스, 제약 조건, 그리고 커넥터가 정의되어야 한다. 이와 같은 요소들을 아키텍처 표현 언어와 연관하여 정의할 수 있다.

3.1.1 추상 클래스

컴포넌트로 정의된다. 컴포넌트는 복합 컴포넌트(composite component)와 원형 컴포넌트(primitive component)로 구분되어 정의된다. 각 컴포넌트의 인터페이스를 ADL의 인터페이스 정의를 통해 나타낸다. 또한 복합 컴포넌트는 구성된 컴포넌트의 종류 및 특성을 정의함으로서 프레임워크 운영시 필요한 정보를 제공한다.

3.1.2 제약 조건

제약조건은 컴포넌트들의 결합에 있어서의 지켜야할 조건을 의미한다. 이러한 결합 관계는 아키텍처를 나타내는 복합 컴포넌트에서 정의된다. 컴포넌트가 다른 컴포넌트의 메소드를 호출하거나 이벤트를 발생하면, 이를 대신 전달하여 주는 매개체인 커넥터가 필요하다. 이 커넥터는 두 개의 컴포넌트 간의 호출 관계를 명확히 하여 호출하는 쪽의 제공 인터페이스와 호출을 받는 쪽의 필요한 인터페이스를 알고 이를 연결시켜주는 역할을 한다. 복합 컴포넌트는 컴포넌트간의 관련성을 나타내며 이러한 관련성을 연결시켜주는 매개체로 커넥터가 정의된다.

따라서 제약조건은 복합 컴포넌트에서 정의된 각 컴포넌트 간의 관련성을 나타냄으로써, 프레임워크를 통해 어플리케이션을 개발할 때 아키텍처 정보로서 사용된다.

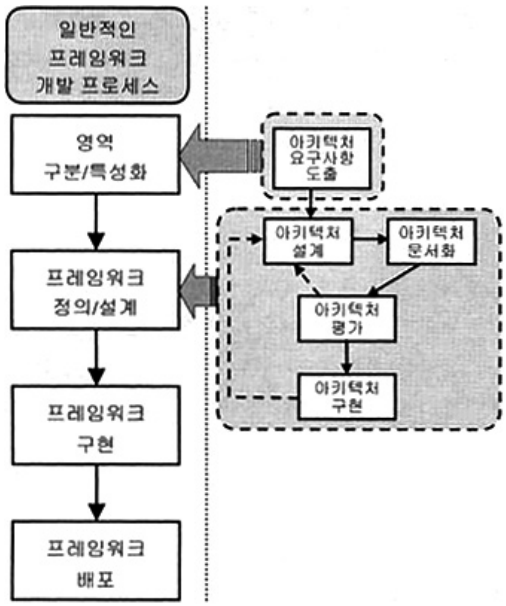
3.1.3 동적 클래스 로딩

동적 클래스 로딩은 추상 클래스와 같이 하나의 인터페이스를 정하여 놓고, 이러한 인터페이스를 제공하는 여러 컴포넌트를 개발함으로써, 원하는 형태의 구현을 선택할 수 있도록 하고자 함이다. 따라서 이와 같은 메카니즘을 구현하기 위해서는 컴포넌트를 개발할 때, 정해진 추상 클래스를 상속하여 구현함으로써 필요로 하는 인터페이스를 제공하도록 하여야 한다. 그러나 이외에 이미 개발이 되어있는 컴포넌트 라이브러리를 사용하여 제공하고자 할 경우에는 기존의 개발된 컴포넌트를 복합 컴포넌트로 랩핑하여 등록함으로써 현재 개

발하는 프레임워크에도 사용할 수 있게 한다.

3.2 아키텍처 기반 프레임워크 개발 프로세스

아키텍처 기반 프레임워크 개발 프로세스는 일반적인 프레임워크 개발 프로세스를 기반으로 한다. 본 논문에서는 이러한 프레임워크 개발 프로세스를 아키텍처를 기반으로 한 프로세스와 혼용하여 아키텍처 개념이 보다 정련된 프레임워크 개발 프로세스로 개선을 한 것이다.



(그림 1) 아키텍처를 적용하기 위한 프레임워크 개발 프로세스

일반적인 프레임워크 개발 프로세스는 기존의 객체 지향 개발 프로세스와 유사하다. 분석, 설계, 구현, 배포의 과정을 거치는 대신 어플리케이션 개발 외의 프레임워크만이 가지고 있는 프레임워크 특성을 개발 프로세스에서 필요로 하고 있는 것이다. 아키텍처 기반 프레임워크 개발 프로세스 역시, 그 기반은 일반적인 프레임워크 개발 프로세스에, 프레임워크를 통해 개발하려는 어플리케이션의 아키텍처 정보를 보다 구체화하여 이 정보를 보다 효율적으로 사용할 수 있도록 하는 것이다. 그러므로 이 개발 프로세스에서는 각 개발 프로세스 별로 아키텍처 정보를 구체화할 수 있도록 필요한 정보를 추출, 설계, 이용, 개발하는 과정을 추가함으로써 보다 개선된 프레임워크 개발 프로세스로 만든 것이다.

3.2.1 문제 영역의 구분 및 특성화

이 단계는 프레임워크의 필요성과 요구사항에 대한 분석을 한다. 이 단계를 통해서는 추상화와 일반적 설계 해결책들이 식별되어야 한다. 또한 아키텍처를 식별하여야 한다. 추상화를 식별함과 동시에 어플리케이션의 전체와 부분적 아키텍처를 식별하여야 한다. 그리고 이러한 부분적 아키텍처와 설계 시 발생할 문제점에 대한 해결책을 식별하여야 한다. 구체적인 영역 분석의 단계의 활동은 다음과 같다.

- 프로세스의 윤곽을 잡는다.
- 기존의 해결책을 조사한다.
- 프레임워크가 프로세스의 어떤 부분을 수행하는지 구분한다.
- 핵심 추상 개념을 구분한다.

이와 같은 과정을 통해서, 시스템의 아키텍처적인 요구사항과 행위적인 요구사항에 대한 Use Case나 다이어그램, 그리고 시나리오 등으로 <표 1>처럼 표현된다.

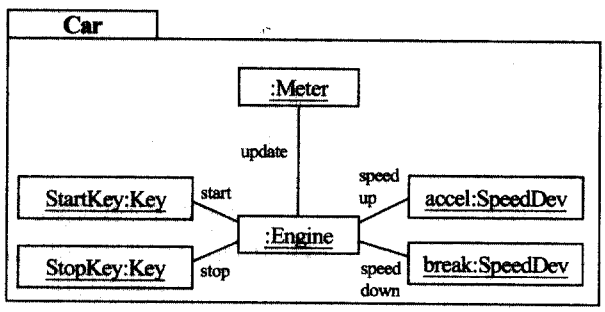
〈표 1〉 추상화된 자동차 Use Case

Use Case	자동차 엔진
Actor	사용자, Car
설명	사용자는 자동차의 엔진의 동작을 조절할 수 있다.
참조기능	R1~R5

이와 같이 프레임워크의 분석활동에서 추출된 어플리케이션들 간의 공통적인 모든 추상화들은 프레임워크로 이전된다. 이를 위해 각 어플리케이션에 대한 정적 객체 모델을 개발해야 한다. 또한 이러한 객체 모델의 모든 객체들은 ADL을 통해 컴포넌트로서 표현된다.

3.2.2 아키텍처 정의 및 설계

이 단계는 영역 분석을 통해 얻어진 추상 개념과 시나리오, Use Case를 통해 소프트웨어 아키텍처를 정의, 설계한다. 일반적으로, 소프트웨어 아키텍처의 구조는 관련성으로 연결된 타입과 같은 노드들의 집합이다. 아키텍처를 기술할 때, 이 노드들은 일반적으로 "컴포넌트"로 나타내고 관련성을 "커넥터"로 나타낸다. 이런 컴포넌트와 커넥터는 "속성"이라고 불리는 다른 정보로서 주석을 달게 한다. 속성은 컴포넌트나 커넥터의 각기 다른 타입들간의 차이점을 명확히 하거나, 다양한 아키텍처적인 분석에 유용한 정보를 제공하기 위해 사용된다.



(그림 4) 자동차 엔진 동작 클래스 다이어그램

위 (그림 2)는 Engine이라는 컴포넌트가 Key라는 컴포넌트로부터 가동되고, 중지된다. Engine의 속도는 SpeedDev라

는 컴포넌트로부터 조종된다. 그리고, Engine의 속도는 Meter 컴포넌트에 의해 표시된다. 이러한 아키텍처를 ADL로서 표현하면 다음과 같다.

〈표 2〉 컴포넌트의 ADL 표현

```
COMPONENT Car
   INTERFACE IS
   END INTERFACE
   IMPLEMENTATION IS
      USES carEngine INTERFACE Engine
      USES carStartKey INTERFACE Key
      ...

      USES speedObserver PROTOCOL Observer

      CONNECT carStarKey.push TO carEngine.startRun
      CONNECT carStopKey.push TO carEngine.endRun
      ...
   END IMPLEMENTATION
END Car
```

〈표 3〉 커넥터의 ADL 표현

```
CONNECTOR Observer
   PROTOCOL IS
   TYPE EventListener
      ROLE changeSourceState IS changeSourceState
      ROLE changeSinkState IS changeSinkState
   END PROTOCOL
   IMPLEMENTATION IS
      BUILTIN
   END IMPLEMENTATION
END Observer
```

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.

ADL을 통해서 컴포넌트와 커넥터를 표현한다. 이 과정에서 컴포넌트의 경우, 각 컴포넌트의 인터페이스를 정의하고, 복합 컴포넌트의 경우, 합성되는 각 컴포넌트간의 관련성을 표현한다.

- 클라이언트의 작업을 보다 쉽게 하기 위해서 도구를 제공한다.
- 반복되는 디자인 패턴을 적용한다.

아키텍처를 설계할 경우 디자인 패턴, 아키텍처적인 패턴 등을 적용하여 설계할 수 있다.

3.2.3 프레임워크 구현

이 과정은 설계 과정을 통해 정의된 아키텍처를 기반으로 하여 전체 컴포넌트 라이브러리를 개발하고, 어플리케이션을 개발을 쉽게 도와주는 지원도구를 개발한다.

- 핵심 클래스를 구현한다.

컴포넌트 기반 프레임워크는 핵심 클래스가 컴포넌트로서 구현된다. 따라서, 추상 클래스에서 정의된 인터페이스를 상속하여, 이를 제공하는 다양한 컴포넌트를 개발하여 구축한다.

- 프레임워크를 테스트한다.

프레임워크를 테스트하는 지표로서 분석 과정에서 만들어진 시나리오, 아키텍처 요구사항 등을 사용한다. 우선 아키텍처 요구사항을 통해 원하는 요구사항을 반영하여 구현하였는지를 확인하고, 정의된 시나리오 순서에 의해 원하는 동작을 하는지를 확인한다.

- 클라이언트에게 프레임워크를 테스트하도록 요청한다.
- 설계의 정련 작업을 반복하고 특징(feature)을 추가한다.

테스트 과정을 통해 미비한 부분을 확인하고, 이를 추가, 수정한다. 아키텍처에 대한 수정의 경우, ADL로 표현되어 있으므로, 해당 ADL을 수정함으로써 아키텍처에 대한 수정이 용이하다.

3.2.4 프레임워크 배급

- 문서를 제공한다.

제공하는 문서로서 일반적인 UML 표현이외에도, ADL을 통한 아키텍처 표현을 제공한다. 특히, 컴포넌트와 아키텍처 등을 선택하는 Hot Spot의 경우 각각의 컴포넌트, 아키텍처에 대한 설명이 필요하다. 각각의 차이점은 무엇이며, 장단점은 어떤 점이 있는지를 문서화를 통해 사용자에게 알려주어야 한다. 그럼으로써 사용자가 원하는 기능을 선택하거나, 정의할 수 있게 되는 것이다.

- 배급에 대한 프로세스를 정한다.
- 클라이언트에 대한 기술적인 지원을 제공한다.
- 프레임워크를 유지 보수하고 갱신한다.

4. 구현 사례

지금까지 아키텍처를 기반으로 한 프레임워크의 구현 방법 및 개념을 제안하였다. 본 4장에서는 3장에서 제안한 방법을 통한 아키텍처 기반 프레임워크를 구현하는 과정과 이 방법을 지원하기 위해 개발된 도구를 설명한다.

4.1 영역 설정

개발될 프레임워크의 영역은 영재교육을 위해 인터넷을 기반으로 학생들간의 협동작업을 가능하게 하는 시스템이다. 이 시스템을 Internet Collaboration System이라 하고 이 프레임워크를 개발하기 위한 컴포넌트 라이브러리가 구축되어 있다. Internet Collaboration System 프레임워크는 서브 시스템으로 다음과 같은 6가지의 기능을 제공해야 한다.

- Chatting 시스템
- BBS 시스템
- White Board 시스템
- Q&A 시스템
- 학생 관리 시스템

회원의 추가, 삭제, 갱신, 열람 기능과 회원의 등급에 따라 열람 정보 및 데이터 접근의 제한 기능, 그리고 조직 및 회원의 통계 자료 추출을 할 수 있는 기능을 제공한다.

본 논문에서는 아키텍처를 적용한 프레임워크의 개발과 운영의 사례로서 위의 서브 시스템 중에서 Chatting 시스템에 대한 경우를 중심으로 개발, 운영하는 과정을 살펴보도록 한다.

4.2 Chatting 시스템

4.2.1 영역 구분 및 특성화

문제 영역의 구분 및 특성화 단계를 통해서 Chatting 시스템에 대한 일반적인 요구사항, Use Case, 아키텍처 요구사항을 추출하였다.

- 기능적 요구사항 및 아키텍처 요구사항 정의

Chatting 시스템을 구성하는 각 기능들에 대한 요구사항의 정의로서 추출된다.

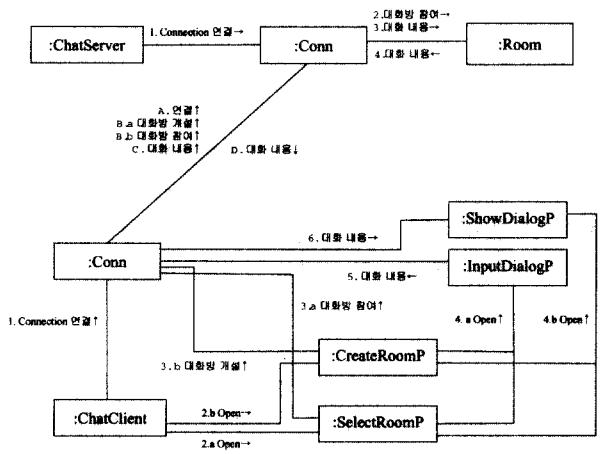
그리고 Chatting 시스템의 아키텍처적 요구사항의 정의로서 Use Case와 같은 형태로서 추출된다.

- Use Case 모델

Chatting 시스템이 요구하는 기능들의 시스템에 대한 정의로서 Use Case를 추출하였다.

- Collaboration 다이어그램

Chatting 시스템을 구성하는 객체들간의 상호작용을 나타내는 것으로, 객체들의 기능에 중점을 두어 표현하였다.



(그림 3) Chatting 시스템의 Collaboration 다이어그램

4.2.2 설계

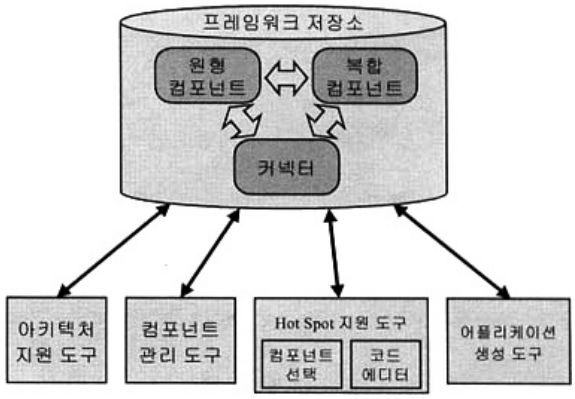
설계 과정을 객체 모델과 ADL을 통한 아키텍처 설계가 이루어진다. 채팅시스템에 대한 각각의 설계 내용으로 객체 모델과 ADL로서 표현한다.

Chatting 시스템 중에서 클라이언트에 해당하는 아키텍처의 경우 UI 컴포넌트에 해당하는 SelectRoomP, ShowDialogP, CreateRoomP, 그리고 InputDialogP와 같이 설계되었다. 그리고 이 외에도 아키텍처 요구사항에 따라 클라이언트/서버 아키텍처에 따른 'Conn' 컴포넌트로 나뉘어져 설계되었다. 이중 채팅 시스템의 클라이언트를 ADL로 표현한 예는 다음과 같다.

```
COMPONENT ChatClient
  INTERFACE IS
    TYPE Module
  END INTERFACE
  IMPLEMENTATION IS
    USES m_selectRoomP INTERFACE groupware.chatting.SelectRoomP
    USES m_createRoom INTERFACE groupware.chatting.CreateRoomP
    USES m_showDialogP INTERFACE groupware.chatting.ShowDialogP
    USES m_inputDialogP INTERFACE groupware.chatting.InputDialogP
    USES m_conn INTERFACE groupware.chatclient.Conn
    USES connector_1 PROTOCOL groupware.chatting.ChattingEventListener
    USES connector_2 PROTOCOL groupware.conn.ConnEventListener
    USES connector_3 PROTOCOL groupware.hookup.HookUp_99051951
    USES connector_4 PROTOCOL groupware.hookup.HookUp_99051952
    CONNECT m_selectRoomP.setRoom TO PC1.Caller
    CONNECT PC1.Definer TO m_conn.sendMsg
    ...
  END IMPLEMENTATION
END ChatClient
```

4.3 아키텍처 기반 프레임워크 지원 도구 개발

아키텍처를 이해하고 이를 기반으로 하여 어플리케이션을 개발하기 위한 지원 도구를 개발한다. (그림 4)는 아키텍처 기반 프레임워크를 위해 구축된 도구의 전체적인 구조와 논리적 개념을 나타내고 있다.



(그림 4) 아키텍처 기반 프레임워크 도구 구조도

이 구조는 일반적인 프레임워크 지원 도구의 형태에 아키텍처 지원 도구를 추가한 형태이지만, 이 도구들의 내부 메커니즘을 살펴보면 기존의 방식에서 약간 다른 면모를 보인다. 도구들이 사용하는 정보가 ADL을 통해 표현된 아키텍처를

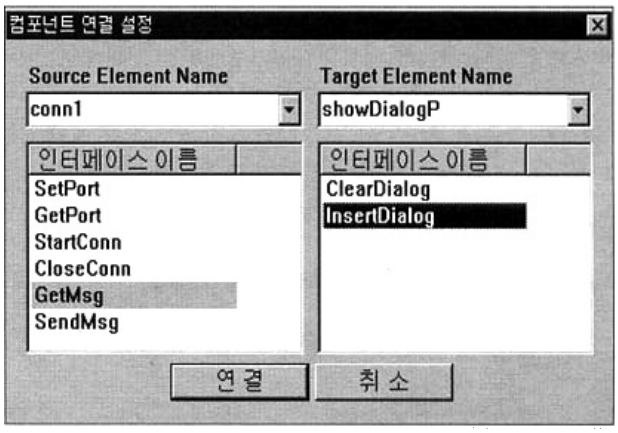
중심으로 사용하여 동작한다는 것이다. 예를 들면 어플리케이션 생성기의 경우, 컴포넌트 컴포지션 시 필요한 아키텍처 정보를 ADL을 통해서 얻고, 이를 기반으로 하여 어플리케이션을 생성한다.

프레임워크 저장소는 아키텍처 정보를 ADL을 통해서 저장하고 해당하는 컴포넌트를 저장한다. 우선 ADL을 통해서 원형 컴포넌트와 복합 컴포넌트, 그리고 커넥터의 모델을 기술하여 저장한다. 그리고 Hot Spot에 해당하는 컴포넌트와 각각 정의된 컴포넌트에 실제 등록된 컴포넌트들의 정보 등을 저장한다.

(그림 5) 프레임워크 저장소 내역 화면

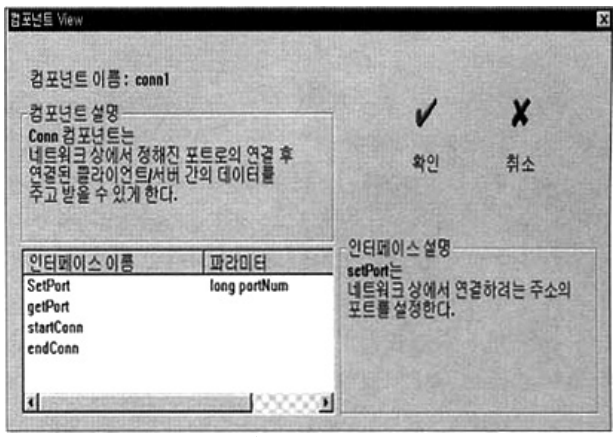
한편, 아키텍처 지원도구에서는 아키텍처 정보를 이해하고 추가, 삭제, 그리고 수정을 위한 도구로 (그림 6), (그림 7)과 같은 화면으로 구성된다.

(그림 6) ADL manager

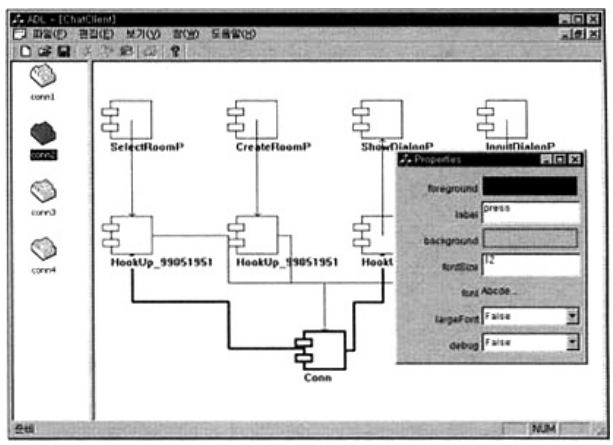


(그림 7) 구성요소 연결 대화상자

Hot Spot 지원도구는 사용자에 의해 정의된 Hot Spot을 선택, 수정할 수 있도록 하기 위한 도구이다. Black-Box 프레임워크의 경우 컴포넌트 선택을 위한 도구이고, White-Box 프레임워크의 경우 소스 코드를 입력할 수 있는 도구이다. 본 논문의 경우 컴포넌트의 초기화에 대한 소스 코드 수정에 관한 사항을 Black-Box에서와 같은 형태로 제공하여 사용자의 편의를 높였다.



(그림 8) 컴포넌트 정보 대화상자



(그림 9) Black-Box 프레임워크 선택 도구

마지막으로 어프리케이션 생성 도구는 Hot Spot을 통해 사용자에 의해 설정된 사항과 기본 템플릿을 기반으로 하여 컴포넌트 컴포지션을 통한 어플리케이션을 생성한다. 이 때, 컴포넌트 컴포지션 시 필요한 아키텍처 정보를 프레임워크 저장소에 저장되어있는 ADL을 통해 얻고 이를 기반으로 하여 어플리케이션을 생성한다.

(그림 10) Web Collaboration 시스템 생성 도구

5. 결 론

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.

참 고 문 헌

[1] Wolfgang Weck, "Independeltly extensible component frameworks," Workshop on Component-Oriented Programming WCOP, 1996.

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

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

[4] Roy H. Campbell, Nayeem Islam, and Peter Madany. Choices, Frameworks and Refinement. Computing Systems, 5(3): 217-257, 1992.

[5] Taligent Inc., Building Object-Oriented Frameworks. Internet : http : //www.ibm.com/Java/education/oobuilding/index.html. A Taligent White Paper, 1994.

[6] David Garlan, Mary Shaw, "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering World Scientific Publishing Co., SingarPore, 1993.

[7] Grady Bookch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, Addison Wesley, 1999.

[8] Robert Allen, David Garlan, "A Formal Basis for Architectural Connection," ACM Transactions on Software Engineering and Methodology, July, 1997.

[9] David Garlan, Dewayne Perry, "Software Architecture: Practice, Potential and Pitfalls," Proceedings of the International Conference on Software Engineering '96, 1996.

[10] Len Bass, Paul Clements, Rick Kazman, Software Architecture in Practice, Addison Wesley, 1998.

이 창 훈

e-mail : lchoon@object.cau.ac.kr

1987년 광운대학교 전자계산학과 이학사

1989년 중앙대학교 전자계산학과 이학석사

1998년 중앙대학교 컴퓨터공학과 공학박사

1994년~현재 중앙대학교 컴퓨터공학과 박사 수료

1999년~2002년 중앙대학교 정보통신연구소 연구전담교수

관심분야 : 소프트웨어 공학, 재사용, 형식 명세 기법, 객체지향 방법론, 프레임워크, 멀티미디어, 컴포넌트 개발방법

이 경 환

e-mail : kwlee@object.cse.cau.ac.kr

1980년 중앙대학교 대학원 응용수학 전공 (이학박사)

1982년~1983년 미국 Auburn대학 객원교수

1986년 서독 Bonn대학 객원교수

1971년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : 소프트웨어 공학, 객체모델링, 재사용 기법, 품질보증, 소프트웨어 표준화