PDM 프레임워크 재 정의 도구 구축
(Construction of a Toolkit for Customizing a PDM Framework)

김 정 아 †
(Jeong Ah Kim)

요 약 PDM(Product Data Management)분야는 일반적인 제조업체에서 제품개발에 필요로 하는 제반정보를 제품기획단계부터 폐기단계까지 제품 생명주기(Lifecyle) 전반에 걸쳐 일원적으로 통합 관리하도록 하는 제품정보 통합관리 솔루션(solution)을 말한다. PDM 어플리케이션은 매우 크고 복잡한 영역이다. 프레임워크의 재사용은 단순한 소스코드나 어플리케이션 일부의 재사용이 아닌 소프트웨어 아키텍처의 재사용을 가능하게 함으로써 생산성 향상을 가능하게 하는 방법이다. 본 연구에서 PDM분야에서 프레임워크를 개발하게된 목적이기도 하다. 즉, PDM 프레임워크는 새로운 PDM 어플리케이션 개발에 드는 시간과 노력을 줄일 수 있게한다. 그러나 프레임워크는 대규모의 클래스들의 집합과 그들간의 복잡한 상호작용을 정의하고 있기 때문에, 프레임워크 기반 재사용 역시 재사용 공정을 지원하는 재사용 환경이 제공되어야한다. 이러한 도구를 바탕으로 큰 규모의 프레임워크를 이해하고 재 정의할 Hot spot을 효과적으로 식별할 수 있게된다. 본 연구에서는 저장소(repository)를 기반으로 컴포넌트와 컴포넌트에 대한 정보를 관리함으로써 컴포넌트를 이해하고 프레임워크에 정의된 hot spot에 새로운 요구사항을 정의하여 새로운 어플리케이션의 개발 과정을 효과적으로 지원하는 도구의 개발하였다.

Abstract PDM is an integrated solution for managing various kinds of document and information for a whole life-cycle of product management. PDM system spans a huge and complex area and requires so many efforts and budgets for development. A framework has been considered a promising way to improve productivity by reusing the software architecture, not just one part of the design or just source code. This was the reason why we developed PDM (Product Data Management) framework. Framework can reduce the time and efforts to develop a new PDM application. However, it also requires supporting environment since a framework is a big set of classes where their interactions are so complex. With this supporting environment, it is easy to understand the framework at a glance and easy to identify what hot spots to be refined to meet new requirements. In this paper, a new framework-supporting reuse environment based on the meta-repository was constructed for easy and convenient reuse.

1. 서 론

프레임워크는 재사용 가능한 설계를 구성하는 상호 연관된 클래스들의 집합과 그들간의 잘 정의된 행위를 정의한다[1]. 프레임워크 내에는 도메인의 범주 안에서 개발될 모든 어플리케이션에 공통적으로 필요한 클래스들을 정의하고 있다. 이런 클래스는 대부분 추상 클래스 개념으로 설계되어 해당 도메인의 중요한 공통의 구조를 표현한다. 또한 프레임워크에는 정의한 추상 클래스들간에 필요한 제어 흐름을 정의해야 한다. 이 제어 흐름에 따라 도메인의 기본 행위 또는 어플리케이션의 수행 및 진행 방법이 결정된다. 프레임워크를 정의 내리면, "프레임워크는 시스템 전부나 일부에 대한 재사용 가능한 요소들이고, 이들 재사용 가능한 요소들은 추상화된 클래스들과 이들의 인스탄스가 상호 동작하는 방식을 함께 정의하고 있는", 클래스 라이브러리보다 큰 규모의 재사용 단위이다[1,2,3,4].

프레임워크를 재사용하여 어플리케이션을 개발하는 것은 프레임워크가 이미 어플리케이션의 행위를 포함하고 있으므로 컴포넌트들의 조합을 정의하는 것이 아니라 어플리케이션의 기본 행위를 구성하게될 컴포넌트를 재 정의 하는 것이다. 재사용 관점에서의 프레임워크는

† 정 회 원 : 관동대학교 컴퓨터교육과 교수
clara@mail.kwandong.ac.kr
논문접수 : 1998년 9월 16일
심사완료 : 2000년 10월 30일

영역 지식, 설계 개념, 이를 바탕으로 한 구현 기술을 포괄적으로 재사용할 수 있다는 점에서 기존의 라이브러리 재사용보다 적극적인 생산성 향상의 방법이다.

프레임워크를 통한 재사용이 개념적으로 라이브러리의 재사용보다 재사용의 효과를 극대화 할 수 있지만, 프레임워크를 기반으로 한 재사용이 반드시 쉬운 작업은 아니다[4]. 프레임워크는 일반적으로 클래스 라이브러리에 비해 규모가 방대하고 복잡한 구조를 지니고 있기 때문에 새로운 어플리케이션 개발에서 재 정의되어야 하는 클래스나 컴포넌트를 식별한 뒤 다른 요소들과의 연관성을 이해한후 클래스를 재 정의 하는 것은 결코 쉬운 일이 아니다. 클래스 라이브러리 재사용을 위한 다양한 지원 환경이 필요한 것과 같이 프레임워크를 이해하고, 재 정의하는 과정을 지원하는 도구가 필요하다. 프레임워크 재 정의를 지원하는 도구 역시 프레임워크를 구성하는 컴포넌트와 이들간의 정의된 상호 작용 관계에 대한 메타 정보를 담은 저장소(Repository)를 바탕으로 이루어져야 한다[5,6].

본 연구는 프레임워크를 기반으로 새로운 어플리케이션 개발을 지원하는 환경을 구축하는 것으로 목표로 한다. 개발하고자 하는 환경의 목표는 프레임워크를 구성하는 실제 컴포넌트와 어플리케이션의 기본 행위를 저장, 관리하면서 프레임워크에 대한 메타 정보를 효과적으로 관리하려는 것이다. 이는 크게 컴포넌트의 구조, 컴포넌트들간의 관련성, 컴포넌트들의 행위 등에 관한 정보를 관리하는 저장소를 관리하는 도구들과 프레임워크 사용자들이 새로운 어플리케이션 개발을 위해 컴포넌트를 이해하고 컴포넌트를 재 정의하는 과정을 지원하는 도구들로 구성한다.

2장에서는 프레임워크의 일반적 정의와 재 정의의 개념을 설명하고 3장에서는 본 논문에서 개발한 재 정의 도구의 개념과 설계에 관하여 설명한다. 4장에서는 개발된 사례와 다른 재 정의 도구들과의 평가를 한후 5장에서 결론을 맺는다.

2. 프레임워크

2.1 프레임워크 기본 개념

프레임워크는 도메인에서의 일반적인 클래스들과 이들간의 연결 관계로 정의되는데, 이런 클래스는 대부분 추상 클래스(abstract class) 개념으로 설계된다. 이 추상 클래스의 목적은 공통성을 추출하기 위해서 존재한다. 이는 프레임워크를 기반으로 새로운 어플리케이션을 개발할 때 이들 추상 클래스를 상속받아서 어플리케이션에 따른 새로운 논리를 만족하는 클래스로 재 정의할 수 있는 여지를 남겨 두는 것이다. 이러한 개념은 프레임워크 설계에서 핫 스팟, 템플리트(template), 후크(Hook) 개념에 해당한다[2,3,4]. 하나의 도메인에 걸쳐 모든 어플리케이션에 공통으로 적용할 수 있는 도메인 공통의 구조를 프로즌 스팟(frozen spot)으로 간주하고 이를 템플리트로 정의한다. 이에 비해 어플리케이션 마다 달라 질 수 있는 구조 또는 논리를 핫 스팟으로 정의하여 후크 메소드로 정의한다. 이런 핫 스팟들은 프레임워크의 확장과 수정을 위해 설계되는 부분이다. 핫 스팟과 템플리트 간의 관련성을 보다 자세하게 정의하면 다음과 같다. 핫 스팟의 개념을 정립하기 위해서 우리는 후크 메소드와 템플리트 메소드를 구분한다. 즉, 후크 메소드 (Hook Method)란 더 복잡한 메소드에 의해 호출될 수 있는 핫 스팟이고, 후크 메소드를 호출해서 다른 작업이 진행되도록 하는 메소드를 템플리트 메소드라고 하고, 객체들 간의 상호작용이나 제어의 일반적 흐름이나 추상화된 행위를 정의한다. 후크 메소드를 정의한 이유는, 상속을 통해서 후크를 재 정의하는 것이 가능하므로 상위 템플리트 메소드의 변경 없이도 객체의 행위를 변경할 수 있기 때문이다. 즉 템플리트 메소드에 정의된 도메인 공통의 구조를 재사용할 수 있게 된다. 후크 메소드를 포함하고 있는 클래스를 후크 클래스(Hook class)라고 정의하고, 템플리트 메소드를 포함하고 있는 클래스를 템플리트 클래스라고 한다[4,7,8].

블랙 박스(Black-box) 프레임워크의 경우는 템플리트 클래스와 어플리케이션 별로 필요한 후크 클래스들, 후크 메소드들을 다 제공함으로써 새로운 어플리케이션 개발이 매우 쉬운 프레임워크인 반면 개발이 어렵다. 이에 반해 화이트 박스(White-box) 프레임워크는 어플리케이션별 구체적인 내용의 개발은 어플리케이션 개발자에게 남겨지는 것으로 상속을 통해 핫 스팟을 채워야 한다. 본 논문에서는 화이트 박스 프레임워크를 대상으로 한다.

2.2 프레임워크 재 정의

클래스 라이브러리의 재사용은 컴포넌트의 조합 방법을 결정하는 것이지만 프레임워크의 재사용은 프레임워크에 정의된 컴포넌트를 상속관계를 통하여 재 정의 하는 것이다. 프레임워크에서의 재 정의는 객체 지향의 상속과 동적 바인딩의 개념을 통해서 구현 가능하다.

프레임워크는 미리 정의된 제어 흐름을 갖고 있기 때문에 어플리케이션 설계자는 어느 특정 부품에만 집중하면 된다. 설계자가 수정한 부품은 프레임워크가 수행되는 동안 프레임워크에 의해 호출되어진다. 동적 바인딩을 통해 객체를 구현과 분리된 대상으로 다룰 수 있

게 한다. 어플리케이션 개발자는 프레임워크를 수정하기 위해 새로운 클래스를 생성해야 하는데, 이는 추상 클래스에 의해 제공되는 기본 틀을 만족하는 코드로 이루어진다.

블랙 박스 프레임워크의 재 정의는 합성과 프레임워크를 구성하는 컴포넌트의 속성(property)을 수정하는 과정이며 화이트 박스 프레임워크의 재 정의는 핫 스팟 정의를 위해 프레임워크에 정의된 추상 클래스를 상속받아 새로운 요구 사항에 대한 내용을 정의하는 과정이다. 물론 블랙 박스 프레임워크에서도 재 정의 및 확장을 위한 추상 클래스등이 제공되어 상속 또는 합성을 통하여 어플리케이션 마다의 클래스를 정의할 수 있도록 지원한다. 단, 화이트 박스 프레임워크와의 차이는 재 정의 및 확장된 클래스들은 프레임워크에 연결하기 위한 객체 합성의 수단이 블랙 박스 프레임워크에는 제공된다는 것이다. 본 연구에서 목표로 하는 프레임워크의 재 정의는 화이트 박스 프레임워크의 재 정의로 상속을 통해 새로운 행위를 정의하거나 기존의 행위를 재 정의 하는 것이다.

2.3 기존의 프레임워크 재 정의 지원 도구

지금까지 프레임워크 재 정의 지원을 지원하는 연구는 많이 이루어지지는 않았다. 그 이유중에 하나는 기존에 개발되어 발표되어진 많은 객체 지향 프레임워크가 Smalltalk 언어를 기반으로 VisualAge/Smalltalk환경에 개발되어 진 연구 결과나 제품이 대부분이다[4,8]. VisualAge/Smalltalk는 객체 지향 개발을 지원하는 IDE(Integrated Development Environment)로써 클래스 브라우저를 통해 재 정의에 필요한 클래스의 식별과, 코드 에디터를 통한 클래스의 편집과, 새로운 클래스와 프로젝트의 생성등을 지원한다. 즉, 프레임워크를 기존의 어플리케이션 프로젝트 처럼 등록 한후 프레임워크의 재 정의에 필요한 기능들을 기존의 어플리케이션 프로젝트 관리하는 방식으로 이용할 수 있다.

이러한 기존의 개발 도구를 이용한 프레임워크의 재 정의 기법과 달리 [9]에서 이루어진 연구에서는 프레임워크의 재 정의를 위해서 저장소를 별도로 관리하고, 저장소에 저장된 내용의 변경을 통해 새로운 어플리케이션이 운영되는 기법을 도입하였다. 이 연구의 특징은 새로운 어플리케이션에 해당하는 코드가 생성되는 것이 아니라, 기존의 프레임워크를 바탕으로 재 정의된 부분을 만족하는 어플리케이션의 실행단을 제공한다. 이는 환경이 지원하는 프레임워크의 특성이 상속에 기반한 개발을 전제로 하지 않고, 프레임워크 자체가 프로세스 또는 엔진의 역할을 수행하는 방식을 택하여 개발하였기 때문이다.

본 연구에서는 [9]에서 연구된 저장소 기반 재 정의 기법의 기본 개념을 바탕으로 한다. 그러나, 저장소에 정의된 프레임워크 모델을 바탕으로 새로운 요구 사항을 만족하기 위해 이 본 연구에서 개발한 도구가 생성하는 코드는 프레임워크에 정의된 클래스의 상속을 통해 이루어지도록 하였다.

2.4 재 정의를 위한 프레임워크의 기본 구조

프레임워크는 도메인에 종속적이므로 한 도메인에서의 설계 개념의 재사용을 강력하게 지원할 수 있으며, 설계 개념 뿐 아니라 개발 산물의 재사용을 위해서 개발 환경에 종속적일 수밖에 없다. 본 연구의 목표는 PDM 프레임워크의 재 정의 지원하는 도구의 개발이므로, [그림1]과 같은 PDM 프레임워크의 개발 환경을 기반으로 한다.

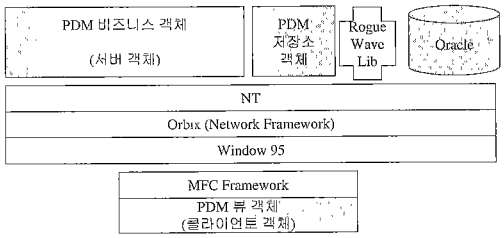


그림 1 PDM 프레임워크의 개발 환경

위와 같은 프레임워크 개발 환경 하에서 개발하는 PDM 프레임워크의 프로그래밍 모델을 [그림 2]와 같이 정의하였다[10,11].

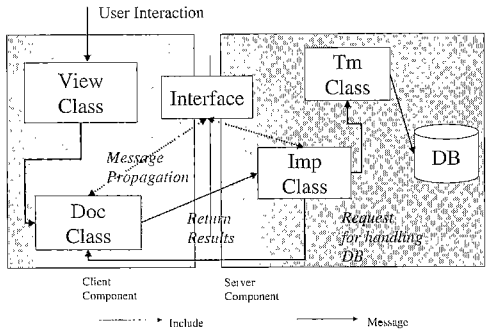


그림 2 프레임워크의 프로그래밍 모델

3. 프레임워크 재 정의 도구 개요

3.1 재 정의 도구 설계 기본 개념

본 연구에서 목표로 하는 도구는 PDM 프레임워크에

서 제공하는 컴포넌트를 재사용하고, 추가적인 PDM 비즈니스 개념을 모델링 하여 이를 프레임워크와 함께 동작할 새로운 컴포넌트로 만들 수 있는 통합 환경이다. 프레임워크의 재 정의는 프레임워크의 기본 행위를 바탕으로 프레임워크를 구성하는 컴포넌트를 상속을 통해 확장 또는 수정함으로써 만들어지는 새로운 어플리케이션은 추가된 컴포넌트의 행위에 의해 처리된다. 프레임워크를 통해 하나의 도메인에서의 기본 소프트웨어 아키텍처를 재사용할 수 있다.

3.1.1 프레임워크 재 정의 대상

[그림 2]에서 프레임워크를 구성하는 프로그래밍 모델을 제안하였다. 제안한 프로그래밍 모델에 따르면 프레임워크의 재 정의 대상을 다음과 같이 구분하였다.

① View 클래스 : 사용자 인터페이스 클래스에 대응되는 MFC 기반 Visual C++ 사용자 인터페이스로 본 프레임워크에서는 MFC의 rc 형태의 resource 화일로 관리하지 않고 MFC의 컨트롤 컴포넌트를 사용하는 클래스이다.

② Doc 클래스 : 사용자 인터페이스에 정의된 하나의 콘트롤과 PDM 도메인에 정의된 비즈니스 객체들간의 대응 관계를 정의한다. 이는 PDM 클래스의 속성과 사용자 인터페이스에 정의된 컴포넌트간의 대응관계와 PDM에 클래스에 정의된 행위와 사용자 인터페이스 정의된 컴포넌트의 이벤트간의 대응 관계를 정의한다.

③ CORBA Interface : IDL 화일을 의미하는 것으로 클라이언트와 서버간의 인터페이스 및 Orbix 객체화하는 데 필요한 정보를 정의한다. 이들 IDL 클래스들 간에도 PDM 비즈니스 객체들간의 상속관계가 그대로 정의되어 있으며 IDL Compiler에 의해 클라이언트 스터브와 서버 스켈레톤을 만들어 낼 수 있다.

④ Imp 클래스 : PDM 비즈니스 객체를 실제로 구현한 클래스이다.

⑤ Tm 클래스 : RogueWave 클래스 라이브러리를 이용하여 데이타 베이스를 관리하는 클래스로 PDM 프레임워크에는 템플리트 클래스 개념의 Tm 클래스 하나가 정의되어 있다.

⑥ DB Schema : 데이타 베이스에 정의될 테이블에 대한 물리적 정의 부분이다.

3.1.2 재 정의 방법

이러한 재 정의 대상을 기반으로 재 정의 가능한 방법을 다음과 같이 정의한다.

① 프레임워크 클라이언트 컴포넌트의 사용자 인터페이스 재 정의 : 클라이언트 컴포넌트 구성 요소 중 View 클래스의 재 정의를 허용한다. 이는 기존의 사용자 인터페이스 콘트롤의 속성 - 크기, 위치, 캡션-의 변경과 새로운 콘트롤로의 대체를 허용한다. 그러나 대체되는 콘트롤은 기존 콘트롤에 정의된 도메인 클래스와의 대응 관계는 그대로 상속받도록 하였다.

② 프레임워크 클라이언트 컴포넌트의 새로운 사용자 인터페이스 정의 : 이는 기존의 View 클래스와 전혀 다른 새로운 사용자 인터페이스를 정의해야 하는 경우로 새로운 사용자 인터페이스 클래스의 추가를 필요로 한다. 이는 View 클래스의 최상의 클래스를 상속받아 필요한 콘트롤에 대한 정보와 도메인 클래스와의 대응 관계는 새롭게 정의하도록 하였다.

③ 도메인 클래스의 재 정의 : PDM 도메인 클래스를 재 정의 하는 것으로 속성의 변경과 인터페이스의 변경 및 행위의 재 정의가 가능하다. 속성의 재 정의는 개념적으로는 속성의 추가, 삭제를 허용한다. 그러나 재 정의는 기본적으로 상속을 기반으로 하는 것이므로 속성의 삭제는 물리적 속성의 삭제가 아닌 속성의 가시성(visibility)만을 변경하는 것으로 처리한다. 도메인 클래스의 오퍼레이션은 인터페이스의 추가, 삭제, 오퍼레이션의 메소드 재 정의를 허용한다. 인터페이스의 삭제 역시 속성의 삭제와 마찬가지로 물리적으로 클래스의 인터페이스에서 제거하지 않고 클래스의 공통 인터페이스(Public interface)에서만 제외하는 것으로 구현한다. 상위 클래스에 정의된 인터페이스를 하위 클래스에서 제거하는 경우 역시, 공통 인터페이스에서만 제외할뿐 인터페이스로는 유지하므로 타입 순응성(Type Conformance)를 보장할 수 있다. 또한 제한적이 나마 새로운 개념의 비즈니스 객체가 추가될 때 이에 해당하는 클래스를 정의할 수 있도록 허용한다.

④ 데이타 베이스 스키마의 변경 : 도메인 클래스의 속성의 정보가 변경됨에 따라서 새로운 속성을 저장하는 데 필요한 필드를 정의할 수 있도록 허용한다. 또한 도메인 클래스에 정의된 속성과 무관하게 데이타 베이스 관리 시스템의 특징을 고려한 필드 속성의 변경도 허용한다.

이와 같은 재 정의 범위는 객체 지향의 클래스 재 정의 방법을 PDM 프레임워크 구성 요소 각각에 적용한 것으로 새로운 클래스의 추가, 기존의 클래스의 확장을 포함한다.

3.2 재 정의를 위한 저장소 설계

프레임워크는 PDM 프레임워크에 정의된 비즈니스 모델, 사용자 인터페이스, 데이타 베이스 모델에 대한 정보를 관리하기 위하여 저장소를 구축하였다. 이 저장소의 정보 역시 객체 모델로 구성된다.

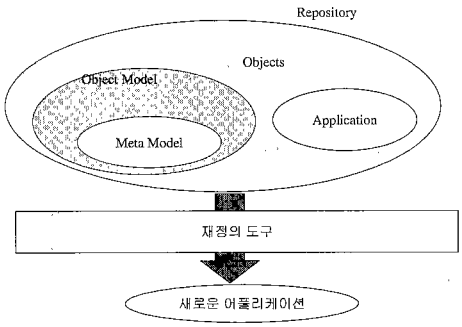


그림 3 메타 저장소의 기본 개념

메타 저장소는 프레임워크가 정의하고 있는 다양한 객체 모델에 대한 메타 정보를 정의하고 있다. 본 연구에서는 프레임워크의 확장성과 재사용성을 보장하기 위하여 프레임워크 개발시에 [그림 2]와 같은 프로그래밍 모델을 적용하였다. 즉, 클라이언트 부분의 사용자 인터페이스와 사용자 인터렉션을 담당하는 클라이언트 프레임워크 클래스와 서버의 비즈니스 논리를 담당하는 서버 프레임워크 클래스 사이의 직접적인 메시지 호출과 같은 종속 관계는 없도록 하였다. 이에 추가 하여, 비즈니스 논리를 담당하는 클래스와 저장소 객체를 담당하는 클래스 사이의 직접적인 메시지 전송이나 호출관계 역시 없도록 프레임워크를 개발하였다. 직접적인 종속 관계 대신, 이들 사이에 매퍼(Mapper) 클래스를 두고 서로 다른 추상화 단계에 정의된 클래스들간의 연결 정보를 정의하고 관리할 수 있는 매퍼 패턴을 적용하였다 [12]. 이 매퍼 패턴은 Mediator 패턴을 서로 다른 추상화 개념에 적용할 수 있도록 변형한 패턴으로 서로 다른 추상화 수준에 정의된 개념들 사이의 직접적인 연결성을 없애고, 이를 매퍼 클래스에 정의하도록 함으로써, 한 클래스의 변경이 다른 클래스에 직접적인 영향을 주지 않도록 하였다.

본 연구에서 구축한 메타 저장소에는 클라이언트 프레임워크와 서버 프레임워크에 정의된 프레임워크 객체 모델 이외에 프레임워크에 정의된 클래스들 사이의 연결성을 처리하는 매퍼 클래스에 대한 모델도 정의하고 있다. 메타 저장소에 정의된 내용을 모델로 정리하면 다음과 같다.

① 비즈니스 모델 : 서버 프레임워크를 정의하는 객체 모델로써, 비즈니스 객체에 대한 클래스 모델과 클래스들 사이의 관련성에 관한 정보를 정의하고 있다. 프레임워크의 핫 스팟과 프로즌 스팟의 구분은 비즈니스 모델에서만 이루어지는 것으로 정의하였다. 비즈니스 모델 중 추상 클래스로 정의된 비즈니스 객체나 가상 함수(Virtual function)으로 정의된 모델의 경우는 프레임워크의 핫 스팟을 저장소에서 관리한다. 이는 재 정의 대상 식별시 저장소에 핫 스팟으로 정의된 부분만을 추출하여 어플리케이션 개발자에게 제공하게 된다.

② 사용자 인터페이스 모델 : 클라이언트 프레임워크를 구성하고 있는 객체 모델로서 사용자 인터페이스 구성에 필요한 정보를 정의하고 있다. UI 클래스별 정보와 UI 구성 요소별 정보로 구분하여 정의하고 있으며, 각각 윈도우 환경에서 UI 객체를 생성하는데 필요한 정보를 저장한다.

③ 저장소 모델 : 지속성(Persistency)을 갖는 비즈니스 객체의 경우 데이타 베이스를 정의하게 되는데, 저장소 생성과 관리에 필요한 기본 정보를 저장한다. 개발한 프레임워크는 데이타 베이스 상호 운영성(interoperability)를 보장하기 위하여 DBTools.h++ 라이브러리를 사용하였기 때문에 데이타 베이스 관리 시스템에 종속적으로 저장소 모델을 정의할 필요가 없이 DBTools.h++ 라이브러리가 제공하는 데이타 베이스 저장 형식을 만족하도록 정보를 정의하여 저장한다.

④ 사용자 인터렉션 모델 : 클라이언트 프레임워크에 정의된 사용자 인터페이스간의 연결 흐름을 정의하는 사용자 행위 (Activity)를 생성하는 데 필요한 정보를 저장한다. 이는 사용자 인터페이스 모델에 정의된 객체들끼리 어떤 이벤트에 의해 상호 호출의 종속 관계를 갖는지를 정의한다. 이 모델을 바탕으로 클라이언트 프레임워크의 행위를 재 정의할 수 있게 된다.

⑤ 클라이언트/서버 매퍼 모델 : 클라이언트 프레임워크에 정의된 사용자 인터페이스 모델과 서버 프레임워크에 정의된 비즈니스 객체들간의 종속 관계를 정의한다. 클라이언트의 요구에 대응하는 서버 객체와 인터페이스간의 대응 관계를 정의한다.

⑥ 비즈니스 객체/저장소 객체 매퍼 모델 : 비즈니스 객체와 저장소 간의 대응 관계를 정의하는 것으로 비즈니스 객체의 속성과 저장소 객체간의 대응 관계를 정의한다.

본 연구에서 구축한 메타 저장소는 기존의 메타 저장소가 담고 있는 객체 모델에 대한 메타 정보에 추가 하여 모델 간의 연결 관계에 대한 매퍼 모델을 추가로 정의하고 있다. 이로써, 프레임워크를 구성하는 한 요소의 변경이 발생할 때, 매퍼 정보를 활용하여 대응되는 객체와의 대응 관계 변경이 필요한지를 도구가 자동 판단하고, 그 변경이 자동으로 반영될 수 있도록 할 수 있게 된다.

3.3 재 정의 도구의 아키텍처

프레임워크 재 정의를 위한 기본 흐름도를 보면 [그림 4]와 같다. 어플리케이션 개발자는 프레임워크의 컴포넌트를 선택하여 기본 행위, 구조 등을 이해하고 재 정의 부분을 식별한 후 컴포넌트 정의 도구를 통해 새로운 컴포넌트를 정의하여 새로운 PDM어플리케이션을 구성하게 된다. 프레임워크에 대한 메타 저장소를 바탕으로 다양한 컴포넌트 정의 도구를 통해 새로운 어플리케이션을 구축하는 것이다.

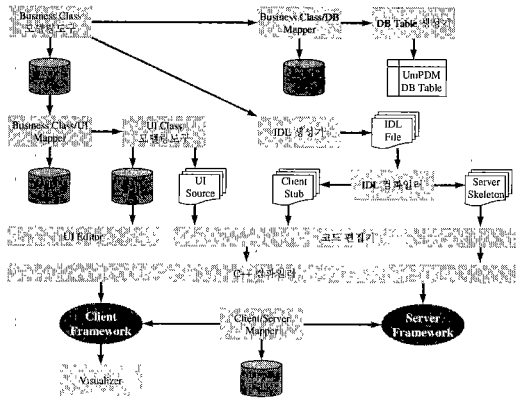


그림 4 프레임워크 재 정의 도구의 기본 구성도

재 정의 도구는 프레임워크에 정의된 CORBA 컴포넌트를 기반으로 비즈니스 논리, 연관된 사용자 인터페이스, 데이타 베이스 저장소를 포함한 어플리케이션을 개발할 수 있도록 지원하는 프레임워크 활용 도구이다.

이 도구의 특성을 정의하면 다음과 같다.

1) 비즈니스 모델링 요소를 확장한다. : 이는 PDM 비즈니스 모델을 기반으로 모델의 확장 및 수정을 통해 전체적인 프레임워크의 재 정의가 이루어진다.

2) 컴포넌트 모델링 도구이며 컴포넌트 생성 도구이다. : 프레임워크를 구성하는 각각의 구성요소인 컴포넌트를 재 정의함으로써 새로운 또는 재 정의된 컴포넌트를 생성하여 추가/변경된 요구사항을 만족하는 어플리케이션을 생성한다.

3) PDM 프레임워크를 기반으로 동작한다. : 재 정의 도구가 생성한 어플리케이션이 동작하기 위해서는 기본 행위를 제공하는 프레임워크를 필요로 한다.

메타 저장소에 저장된 모델들로부터 다음의 프레임워크 재 정의 과정을 거쳐 새로운 PDM 어플리케이션을 개발하게 된다. 그러나 이러한 재 정의 과정은 엄격한 순서를 정의하고 있는 것은 아니다.

(1) 비즈니스 객체 모델의 확장 및 재 정의 : 새로운 PDM 비즈니스 객체가 등장하거나 프레임워크에 정의된 PDM 객체의 행위 및 구조를 변경해야 할 경우 비즈니스 객체 모델을 재구성할 수 있다. 새로운 어플리케이션 개발자는 상속 받을 기존의 추상 클래스를 선택하고, 새로운 클래스를 구성하여, 필요한 속성을 추가하거나 행위를 재 정의함으로써 새로운 비즈니스 프로세스를 추가할 수 있다. 이 과정에서 비즈니스 객체 모델의 변경은 이와 관련된 저장소 모델과 사용자 인터페이스의 변경을 유발한다. 추가적인 변경이 필요한 지의 여부는 저장소에 정의된 매퍼 모델을 분석하여 도구가 판단한다. 기존의 비즈니스 객체를 수정하는 경우 경우는 도구를 통하여 사용자 인터페이스와 저장소 모델에 자동 반영된다. 그러나 추가의 경우, 저장소에 해당하는 테이블이나 필드는 도구에서 자동으로 생성해주며, 사용자 인터페이스의 경우는 사용자 인터페이스 편집기를 통해 해당하는 요소의 추가후 비즈니스 모델과의 대응성만 정의하면 된다. 새로운 속성의 추가의 경우는 저장소에 필요한 필드 및 새로운 모델의 추가가 필요하며 더불어 사용자 인터페이스의 속성 추가가 필요하다. 행위 추가의 경우는 필요한 사용자 인터페이스 요소의 추가만이 필요하다. 행위 추가시에 필요한 사용자 인터페이스 요소의 형태를 정의하면 사용자 인터페이스 생성기에 의해 새로운 요소가 생성된 후 대응되는 화면에 추가된다.

(2) 사용자 인터페이스의 재 정의 : 프레임워크에서는 모든 것을 객체로 간주한다. 사용자 인터페이스 역시 하나의 객체로 인터페이스 외관에 해당하는 구조와 인터페이스의 행위를 재 정의할 수 있다. 사용자 인터페이스 편집기를 이용하여 어플리케이션 개발자가 기존의 인터페이스의 대치나 새로운 인터페이스의 추가 및 삭제 할 수 있다. 단, 새로운 요소의 추가의 경우는 비즈니스 객체와의 대응 관계를 선택해 주면, 지원 도구에서 필요한 매퍼 모델을 생성하여 처리한다. 비즈니스 객체에서 속성의 삭제나 인터페이스의 삭제 없이 사용자 인터페이스의 삭제가 이루어질 경우는 화면상에서만 보이지 않게 하며, 서버로 전달시 속성의 경우는 기본값을 전달하도록 코드를 생성한다.

(3) 저장소 모델의 재 정의 : 저장소 모델의 경우 비즈니스 객체 모델의 재 정의에 의해 도구에서 자동 재 정의될 수 있지만, 새로 정의된 저장소 모델에 대한 상세한 내용을 재 정의 할 필요가 있다면 저장소 편집기를 이용하여 세밀한 속성의 수정이 가능하다.

(4) 컴포넌트 이해 및 재 정의 대상 선정 : 프레임워크를 구성하는 컴포넌트별로 컴포넌트의 구조와 행위를

이해함으로써 프레임워크가 제공하는 기본 행위를 새로운 요구 사항에 맞도록 재 정의해야 하는 요소를 식별하게 된다. 이는 프레임워크를 구성하는 각 컴포넌트를 독립적인 단위로 저장하여, [6]에서 제공한 Runtime Object Library를 사용하여, 어플리케이션을 실행시키지 않고도, 각 컴포넌트의 동작 방식을 이해할 수 있도록 지원한다.

이러한 과정과 재 정의 도구와의 관계는 [그림 5]와 같다.

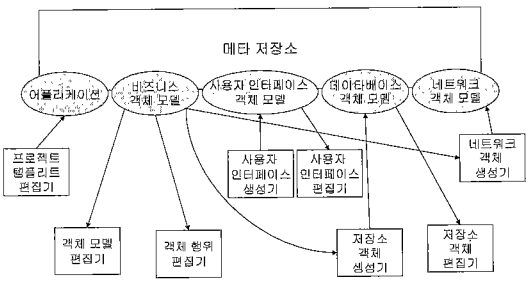


그림 5 메타저장소와 재 정의 도구간의 관계

도메인 객체는 비즈니스 클래스 모델링 도구로부터 생성된 모델을 정의하고 모델링 도구가 생성한 IDL 화일과 연결한다. 비즈니스 클래스 모델링 도구에서는 기본적으로 UML(Unified Modeling Language)[13]를 기반으로 모델을 작성한다. 사용자 인터페이스 객체는 사용자 인터페이스 생성기에 의해서 생성된 화면 단위들로 화면을 구성하는 사용자 인터페이스 구성 요소들을 포함하고 있다. 도메인 객체에 정의된 속성이나 오퍼레이션은 사용자의 조작을 위해서 사용자 인터페이스에 표시되게 마련이다. 즉, 속성의 단순한 입출력의 경우는 화면에 표현된 속성 정보를 직접 조작함으로써 이루어진다. 예를 들어 텍스트 박스에 직접 입력하거나 값을 텍스트 박스에 나타내는 작업을 통해 간단한 속성에 대한 조작이 이루어진다. 또한 복잡한 논리를 갖는 오퍼레이션의 경우도 일단 사용자의 선택을 통한 메시지의 발생을 위해 사용자 인터페이스에 구성 요소로 표현되기 마련이다. 데이타베이스 객체는 어플리케이션이 영구 저장해야 하는 객체에 대한 상태를 저장하기 위한 것으로 속성들의 값을 저장하기 위한 데이타 베이스 테이블로 저장한다.

3.4 프레임워크 재 정의 도구 설계

재 정의 도구의 사용자(Actor)는 크게 프레임워크 구축자와 프레임워크를 재 정의하여 새로운 어플리케이션을 개발하는 사용자 그룹으로 구분할 수 있다. 프레임워크 구축자는 프레임워크에 대한 메타 저장소 구축을 위한 다양한 정보를 입력하는 사람이고 어플리케이션 구축자는 메타 저장소의 내용을 기반으로 새로운 어플리케이션 요구 사항에 부합하는 내용을 재 정의하는 사용자 그룹이다.

현재, 프레임워크를 바탕으로 어플리케이션 생성시, 프레임워크의 저장소와 별도로 어플리케이션을 위한 저장소를 구축하고, 재 정의된 부분을 별도로 관리하고 있으며, 이 저장소를 통해 계속적인 어플리케이션의 재 정의가 이루어지도록 하였다.

1) 시각화 도구를 통한 프레임워크의 이해

시각화 도구를 통하여 어플리케이션 개발자들은 프레임워크에 정의된 시스템의 기본 행위를 이해한다. 또한 프레임워크의 기본 구조를 이해하기 위해 클라이언트 컴포넌트 구조 및 각 컴포넌트의 행위를 이해한다. 서버 컴포넌트의 클래스 계층도와 각 서버 컴포넌트의 구성 요소들을 이해할 수 있게 된다. 이 과정의 목적은 프레임워크를 구성하는 컴포넌트 중 어느 부분이 새로 개발해야 하는 어플리케이션과 기능적인 면에서 일치하지 않은지를 식별하려는 것이다.

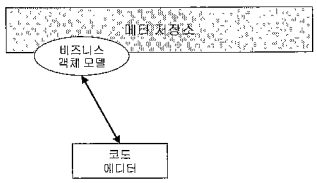
2) 사용자 인터페이스 재 정의

사용자 인터페이스 재 정의 경우는 다시 크게 두 가지 재 정의로 구분할 수 있다. 첫째는 기존의 사용자 인터페이스 컴포넌트를 그대로 유지한 채로 사용자 인터페이스의 구성 요소의 타입을 변경하는 경우이다. 이는 사용자 인터페이스 편집기를 통하여 변경하면 이를 메타 저장소의 사용자 인터페이스 객체 모델에 반영한다. 사용자 인터페이스가 추가되거나 새로운 사용자 인터페이스 요소의 추가는 먼저 비즈니스 모델러를 통해 기존의 사용자 인터페이스 클래스의 수정을 통해 새로운 요소를 추가하고 이와 도메인 객체간의 관련성을 정의한 후 이를 바탕으로 자동으로 생성된 사용자 인터페이스 화면을 편집한다.

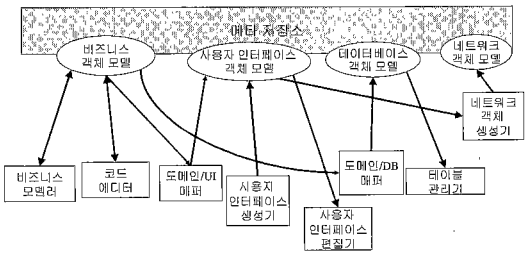
3) 새로운 도메인 객체의 추가 및 기존 도메인 객체의 재 정의

가장 기본적인 도메인 객체의 재 정의는 기존의 구현을 수정하여 새로운 개념의 어플리케이션 객체를 구현하는 것이다. 이는 코드 에디터를 통해 기존의 서버 객체의 인터페이스에 대한 새로운 구현부의 추가를 통해 이루어진다. 이는 해당하는 비즈니스 객체 모델의 메타 저장소에 반영된다. 그러나 새로운 인터페이스의 추가나 완전히 새로운 도메인 객체의 추가의 경우는 프레임워크의 재 정의는 복잡해진다. 먼저 비즈니스 객체 모델러

를 이용하여 새로운 인터페이스 및 새로운 도메인 개념을 모델링하고, 이 객체에 필요한 사용자 인터페이스 클래스와 데이타 베이스 클래스를 정의한다. 이 과정에서 도메인 객체와 사용자 인터페이스, 도메인 객체와 데이타 베이스 클래스간의 대응 관계를 정의해야 한다. 이를 바탕으로 재 정의 도구가 제공하는 IDL 생성기를 통해 네트워크 객체와 매핑 정보를 생성한다.



(a) 객체 행위의 재 정의



(b) 새로운 비즈니스 클래스 추가

그림 6 비즈니스 객체 재 정의 과정과 메타 저장소

프레임워크의 재 정의는 저장소에 정의된 프레임워크 모델을 바탕으로 핫 스팟들의 재 정의만을 허용한다.

4. 재 정의 도구의 구현 및 평가

4.1 재 정의 도구 모듈

재 정의 도구는 [그림 5]의 모듈 구성도에 따라 크게 5개의 서브 시스템으로 개발하였다. 지원하는 PDM 프레임워크와 동일한 운영 환경에서 개발하였으며, 저장소는 Window NT, 오라클을 기반으로 구현하였으며 저장소를 통한 재 정의 클라이언트는 Window 9x, Visual C++ 프로그래밍 환경을 지원하도록 개발하였다. 각각의 서브 시스템의 구현된 기능은 다음과 같다.

1) 도메인 클래스 생성기/관리기 : 프레임워크를 구성할 Model 클래스, View 클래스, Persistent 클래스등의 프레임워크 컴포넌트 클래스들을 생성하고 이들에 대한 메타 정보 및 매핑 정보를 관리한다. 프레임워크를 구성하는 도메인 클래스들의 계층 구조와 상호 연결 관계를 파악하고 있어 프레임워크 재 정의한 결과를 바탕으로 새로운 어플리케이션 코드를 생성한다.

2) 사용자 인터페이스 클래스 생성기/관리기 : 본 연구 대상의 프레임워크는 실제 어플리케이션을 개발할 수 있는 사용자 인터페이스 컴포넌트까지 포함한 것으로 가정하였다. UI(User Interface) 관리 서브 시스템은 도메인 클래스를 서버 객체인 도메인 클래스와 클라이언트의 사용자 인터페이스 클래스와의 연결성을 관리한다. 도메인 클래스에서의 변경은 저장소에 저장한 매핑 관계를 통해서 자동으로 사용자 인터페이스 클래스에 반영시킨다. 또한 사용자 인터페이스는 기본적으로 사용자들의 편이성을 고려하여 빈번한 수정 보완될 수 있는 요소로 UI 관리 시스템은 Visual C++를 이용한 화면의 생성 및 편집을 지원한다.

3) IDL 클래스 생성기 : 본 연구에서 대상으로 하는 프레임워크는 CORBA상에서 운영되는 컴포넌트이고 객체들이므로 도메인 클래스의 경우는 CORBA객체로의 생성 및 기존의 CORBA 객체와의 매핑 관계를 관리하여야 한다. IDL 클래스 생성기는 도메인 클래스 생성기에 의해 정의되는 클래스들 중 CORBA객체로 지정되면 이를 IDL로 생성하고 CORBA 객체와의 매핑 정보를 생성 관리하게 된다.

4) 영속적 클래스 생성기 : 도메인 클래스 중 데이타 베이스에 저장될 필요가 있는 경우 영속적 클래스를 정의하고 이 클래스에서 데이타 베이스 관련 처리를 담당하도록 하는 것이 일반적인 프레임워크 구현 방법이다. 영속적 클래스 생성기는 도메인 클래스와 영속적 클래스간의 대응 관계를 정의함으로써 도메인 클래스의 변경에 따른 영속적 클래스의 변경을 담당하게 된다.

5) 프레임워크 메타 정보 저장소 설계 및 구현 : 각 도구들로부터 생성된 프레임워크 기본 클래스들에 대한 메타 정보와 클래스들간의 매핑 정보 및 참조 관계에 대한 메타 정보를 관리하는 저장소이다.

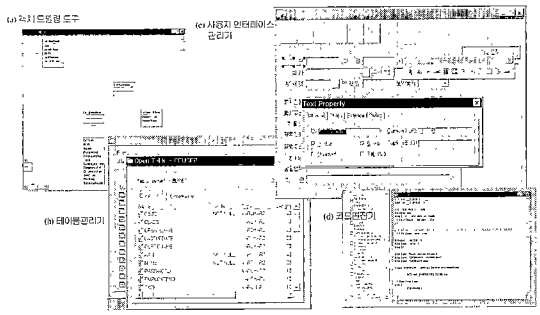


그림 7 서브 시스템별 구현 예

이들 서브 시스템의 실제 구현 결과 PDM 재 정의에 활용되는 실행 예는 [그림 7]과 같다.

4.2 평가

본 논문에서 개발한 재 정의 도구의 기능성을 평가하기 위해서 PDM프레임워크를 지원하는 재 정의 도구들과 비교하였다. 평가 결과는 [표 1]에 제시하였다. 비교의 대상은 PDM 프레임워크로 널리 사용되고 있는 Metaphase, WorkManager, Matrix이며[14,15,16]을 선정하였다. 이들 프레임워크는 이미 상용화된 대형 프레임워크로써 일부 객체 지향 개념을 제공하고 있으며, 각 프레임워크의 재 정의를 지원하는 간단한 기법과 도구를 제공하고 있다. 대부분 기존의 사용자 인터페이스 생성기와의 연계를 통해 자신들의 비즈니스 모델에 대한 새로운 사용자 인터페이스 생성을 지원한다. 비즈니스 모델의 변경의 경우는 API를 통한 변경을 지원하는 것이 대부분이다. 또한 저장소 모델의 경우는 어플리케이션 개발자의 노력에 의해 재 정의된다. 그러나 이들 도구는 수년에 걸쳐 많은 기업에서 활용되어온 입증된 프레임워크이므로 특별한 재 정의 도구 보다는 API를 통한 재 정의를 지원한다. 그러므로 평가가 도구 자체의 성능 비교이기 보다는 각 프레임워크가 제공하는 재 정의 기능을 본 연구가 개발한 지원 환경이 만족하는지를 평가하였다. 각 프레임워크가 제공하는 기능의 경우는 [5]를 참조하여 추출하였다. 또한 각 프레임워크에서 정의된 재 정의 기법과 지원 도구가 있다면 그 명칭을 제시하였다. 평가 결과 본 연구에서 개발한 재 정의 도구 역시 재 정의에 필요한 대부분의 기능을 제공하고 있는 것을 알 수 있으며, 저장소 기능과 모델간의 매핑 정보 관리를 통해, 모델 하나의 변경으로 인한 재 정의의 경우, 매핑 정보를 통해 새로운 어플리케이션 생성이 더 용이 한 것으로 평가 할 수 있다.

표 1 프레임워크 재 정의 도구 평가 표

분 류	개발한 도구가 제공하는 기능 항목	WorkManager	Metaphase	Matrix
비지니스 모델러	비지니스클래스정의 Class 생성 가능	0 Class생성 가능	◎ MODeL로 정의	◎
	관련성정의 상속,연결,집합관계 정의 기능	0 Link 속성을 이용하여 Relationship정의 기능	◎ MODeL로 정의	◎ 상속관계 지원
	비지니스모델관리 저장,조회,출력,편집 행위 Viewer 제공	0 저장,조회,출력,편집 기능	◎ MODeL Viewer제공	◎
	Component Repository 등록	◎ 재사용 가능	◎ Migration Tool제공	◎
UI Builder	UI 컴포넌트 정의 다양한 UI 컴포넌트 제공	◎ Template Editor를 이용 (Classification 모듈)	◎ ICE Browser제공	0 Form이나 Report로 정의, 제한적
	스크립트언어 정의 및 변환 - X	◎ Macro API를 지원	◎ IML제공	◎ Form이나 Report가 객체로 정의/관리됨
	리소스변환 리소스컴파일러 제공	×	× GUI Server제공	× 제공하지 않음
	파일관리 저장,복사,인쇄,열기 기능	◎ Template Editor를 이용 저장,복사,인쇄,연기 기능	◎ ICE Browser제공	◎ 객체로서 관리
UI Builder	Component Repository 등록	◎ Template Editor를 이용	◎ Migration Tool제공	◎
코드편집기 기능	코드편집 코드 편집기 제공	0 Macro API를 이용 / Editor로 편집 기능	◎ vi 에디터 이용	◎ MQL module에서 제공
	코드버전관리 코드 check-in/out	×	◎ 소스코드Check-In/Out	×
	UI/DB 코드편집 대피 제공	0 Macro API를 이용 / Editor로 편집 기능	◎ vi 에디터 이용	0
	패턴관리 ..가의 코드 패턴 제공	△	0 Hot Spot구분없음	◎
Visualizer	Trace ×	0 Trace Command를 통해 해당 Object history 지원	◎ IML Debugger 제공	0 verbose on 기능
	데이타계측 ×	×	◎ IML Debugger 제공	×
UI/DB Mapper	Mapping 매퍼 제공	0 Template Editor에서 정의된 Class간의 Mapping	0 MODeL로 정의	0 Form/Report의 Field와 객체의 속성을 Mapping
	Mapping관리 저장소에 별도 관리	0 Template Editor, Class간의 Mapping 관리	0 MODeL로 정의	0
	Wrapping X	△ 정의된 Class 속성변경으로 가능	△ IML데이터타입만 지원	△
	화면출력 UI 컴포넌트 생위 시각화기 제공	△ Template Editor에서 UI Test기능	0 UI 컴포넌트,API 지원	△
메세지/UI Mapper	Mapping 매퍼 제공	0 Template Editor에서 정의된 Class지정	◎ MODeL로 정의	△ UI Component와 객체행위는 연결시킬 수는 없음
	Event Handler X	0 Template를 통한 Event Message를 Macro로 조정	◎ Predefined Event제공	0 객체의 Event에 Event Handler를 정의할 수 있음
	Mapping관리 저장소에 별도 관리	0 Macro API를 통해 기능	◎ MODeL로 정의	△

■범례

- × : 기능을 제공하지 않음
- △ : 구현가능하나 복잡하고 쉽지 않음
- 0 : Customizing하여 쉽게 구현가능함
- ◎ : 우수함(조금만 Customizing하여 그대로 사용가능함)

5. 결 론

객체 지향 프레임워크는 새로운 소프트웨어 개발 방법으로 재사용을 극대화 할 수 있다. 그러나 프레임워크는 일반 어플리케이션을 개발하는 것 보다 더 많은 비용과 노력을 투자해야 한다. 프레임워크는 어플리케이션의 설계 개념과 이에 바탕 한 실제 개발 산물을 실질적인 컴포넌트 형태로 재사용함으로써 보다 확실한 생산성 향상을 보장할 수 있다. 라이브러리를 이해하고 라이브러리를 통합하는 것과는 달리 어플리케이션 전체 개념을 이해하고 주어진 새로운 요구 사항과의 차이점을 식별하여 해당 컴포넌트를 상속에 의해 재 정의하면 된다. 어플리케이션 전체의 재사용을 통한 여러 이점을 얻을 수 있지만 재사용의 대상이 방대함으로 인해 쉽게 프레임워크를 이해하고 이를 재 정의할 수 없다. 프레임워크의 재사용을 지원하기 위해서는 기존의 검색이 중요시 되어온 라이브러리 지원 도구와는 다른 환경의 개발이 필요하다. 본 연구에서는 프레임워크에 대한 메타 저장소를 바탕으로 프레임워크 재 정의 지원 도구를 설계, 개발하였다. 재 정의 지원도구를 통해 프레임워크의 기본 행위를 이해하고, 프레임워크 구성 컴포넌트를 이해하며 재 정의가 필요한 요소를 식별하여 이들의 객체 모델, 사용자 인터페이스 모델, 저장소 모델을 하나의

환경에서 재 정의할 수 있도록 하였다. 이로써 해당 도메인에서 어플리케이션 개발 경험이 부족한 개발자들도 짧은 교육 과정을 거쳐 새로운 어플리케이션을 개발할 수 있다. 또한 하나의 환경 하에 부품 모델링 및 부품 생성을 통한 어플리케이션의 개발이 이루어질 수 있다는 장점을 갖는다. 현재 재 정의는 저장소에 정의된 모델의 재 정의를 통해서만 이루어지므로, 모델링 도구가 가지고 있는 의미 검사 및 타입 검사 수준에서 아키텍처와의 부합성을 검증하고 있다. 추후 모델의 재 정의가 PDM 프레임워크의 전체 아키텍춰 관점에서 일관성을 유지하고 있는지에 대한 모델 검사기의 기능이 보강되어야 한다.

참 고 문 헌

[1] Pree, W., Framework Patterns, SIGS Books, New your, NY, 1996

[2] Pree,. W., Design Patterns for Object-Oriented Software Development, Addison-Wesley /ACM Press, Reading, MA, 1995

[3] Gamma, E., Helm, R, Johnson, R., and Vlissides, J. Design Patterns : Elements of Reusable Object-Oriented Software, Addison-Wesley/ACM Press, MA, 1995.

[4] R. Johnson, Frameworks = (Components + Patterns) , CACM Vol. 40., No. 10, Oct., 1997, pp 39-42

[5] CIMData, PDM Buyer's Guide, 1994, pp 5 - 54

[6] Ed Metak, Jean Caputo, Dynamic Runtime Objects : Building Applications Your Users Can Modify at Runtime, Microsoft Systems Journal, July, 1997, pp 49-74

[7] Mike Potel, MVP : Model-View-Presenter, The Taligent Programming Model for C++ and Java, Technical Report of Taligent, 1996

[8] Paul Dustin Keefer, An Object Oriented Framework for Accounting Systems, MS Thesis of University of Illinois at Urbana-Champaign, 1994

[9] Martine Devos, Michel Tilman, A Repository-based framework for evolutionary software development, Technical Report, AF1179-1

[10] Michael Mattsson, Object-Oriented Frameworks : A Survey of methodological issues, LU-CS-TR: 96-167

[11] Rational, Object-Oriented Analysis and Design Using UML : Professional Services, Notes, Rational Software Corporation, 1997

[12] Jeong Ah Kim, "Abstraction Strategies for Object-Oriented Framework," in Proceedings of AOM conference, 199913. Hans Albrecht Schmid, Design patterns for constructing the hot spots of a manufacturing framework, JOOP June, 1996, pp 25 - 37

[14] Jeong Ah Kim, Jin Hong Kim ,Nam Kyu Park, Development of PDM framework and its customization environment, TOOLS Pacific28, 1998, pp.45-54

[15] HP Metapahase : Object Management Framework, PDM Manual

[16] PDM Enablers : Joint Proposal to the OMG in Response to OMG Manufacturing Domain Task Force RFP1, mfg/98-02-02, OMG

김 정 아

1990년 중앙대학교 컴퓨터공학과 대학원 졸업(공학석사). 1994년 중앙대학교 컴퓨터공학과 대학원 졸업(공학박사). 1994년 ~ 1996년 중앙대학교 기술과학연구소 객원연구원. 1996년 ~ 현재 관동대학교 컴퓨터교육과 부교수. 관심분야는 객체지향방법론, 컴포넌트개발 방법론, 소프트웨어 재사용, 소프트웨어 품질개선