플랫폼 독립적 컴포넌트 기반 개발을 위한 XML-SOAP 활용 객체지향프레임워크 SOAF

(An Object-oriented Framework SOAF utilizing XML-SOAP for Platform-Independent Component-Based Development)

장 진 영 † 최 용 선 ††
(Jinyoung Jang) (Yongsun Choi)

요 약 최근 대부분의 대규모 기업정보시스템은 기능재활용성, 다종의 시스템 리소스, 다중 플랫폼 등을 지원하기 위해 다층구조의 미들웨어 또는 프레임워크를 기반으로 하고 있다. 그러나 이러한 다층 및 다중 플랫폼 분산 구조는 미들웨어 간의 컴포넌트 및 메타정보에 대한 상호운용성 문제를 제기한다. 본 논문은 추상화 프로그래밍 스타일과 XML-SOAP에 기반한 컴포넌트 보존 방법을 통해서, 다종의 리소스를 지원하고 플랫폼에 독립적인 컴포넌트 기반 개발을 가능케 하는 객체지향프레임워크 *SOAF (Simple Object Application Framework)*을 제시하고 그 아키텍쳐 및 주요 특징에 대해 소개한다.

키워드 : 객체 지향 설계, 프레임워크, 패턴, XML, SOAP, 컴포넌트 기반 개발, 적응형 객체 모델

Abstract Recently, large-scale enterprise information systems are commonly based on the multi-tiered middleware or frameworks to support such requirements as functional reuse, heterogeneous system resources, and multiple platforms. However, these multi-tiered or distributed multi-platform architecture incurs the interoperability issue of the components and metadata among the middleware. This paper introduces the *Simple Object Application Framework* (*SOAF*) which supports heterogeneous resources and platform-independent component-based development, with the abstract programming style of the object-oriented frameworks and the XML-SOAP based component persistence mechanism.

Key words : Object-Oriented Design (OOD), Framework, Pattern, XML, SOAP, Component-Based Development (CBD), Adaptive Object Models

1. 서 론

객체지향프레임워크 (Object-Oriented Application Framework)는 재사용할 수 있는 소프트웨어 컴포넌트와 반완성 상태의 어플리케이션 골격을 제공하여, 어플리케이션 개발에 있어서 분석 및 설계와 코드 전반에 걸쳐 재사용 기술의 효과를 극대화한다[1]. 이 같은 장점을 바탕으로 가트너그룹은 2000년대 초반에 개발되는 어플리케이션의 70%는 컴포넌트 및 객체지향프레임워크 기반으로 구축될 것으로 전망한 바 있다[2]. 기업어플리케이션 개발에 적용할 수 있는 IBM의 *Sanfrancisco*[3] 또는 자바기반 웹-데이타베이스 어플리케이션 구축을 위한 *Expresso*[2] 등은 이와 같은 객체지향프레임워크 개발노력의 예로 들 수 있다.

그러나 기존의 프레임워크들은 최근의 정보시스템 환경에 요구되는 다종의 시스템 리소스에 대한 지원과 다중 플랫폼 간 컴포넌트 및 메타정보에 대한 상호운용성의 문제를 해결하기에는 부족한 측면이 있다. 최근에 제시된 *웹서비스*와 같은 *서비스지향아키텍쳐(Service-Oriented Architecture)*[4]는 서비스에 대한 묘사정보인 WSDL과 상호운용성을 보장하는 RPC 메커니즘인 SOAP 등을 통해 필요한 서비스를 상호운용적인 방법으로 컴포넌트화 할 수 있게 함으로써, 이와 같은 문제에 대한 해결방안의 하나가 될 수도 있다. 하지만 컴포넌트의 로직을 서버측에서 실행하고 결과를 돌려받기위한 잦은 서비스 호출은 네트워크 홉을 과다사용하게 되는 단점이 있어, 주로 높은 입도(granularity)로 설계되어 활용되도록 권고되고 있다[4,5]. 즉, 많은 수의 시스템 리소스를 웹서

† 비 회 원 : (주)현세시스템 연구원
pongsor2@hotmail.com
†† 정 회 원 : 인제대학교 시스템경영공학과 교수
yschoi@inje.ac.kr
논문접수 : 2003년 10월 14일
심사완료 : 2004년 6월 17일

비스 등으로 컴포넌트화 하는데에는 한계가 있으며, UI 정책과 같은 클라이언트에서 실행되어야 하는 로직을 컴포넌트화하는 것은 원천적으로 불가능하다.

본 논문에서는, 다종 리소스 지원과 다중 플랫폼상에서의 컴포넌트 기반 개발을 통해 효율적인 기업 어플리케이션구축을 지원하도록 설계된 XML-SOAP 활용 객체지향프레임워크 SOAF(Simple Object Application Framework)의 구조와 활용방안에 대하여 논의한다.

2. SOAF의 문제영역 및 관련연구

본 장에서는 다종 리소스 및 다중 플랫폼의 지원 등 SOAF의 대상이 되는 문제영역과 이를 해결하기 위해 SOAF에 적용된 관련연구들에 대해 소개한다.

2.1 SOAF의 문제영역

최근의 기업용 어플리케이션은 데이타베이스, XML, Persistence, 웹 인터페이스, 분산객체, 리소스풀링, 로깅 등 다양한 고난도의 기술들을 요구하고 있다. 특히 다종의 시스템 리소스들이 지속적으로 추가 개발되어가고 있으며, 이에 따라 이를 통합하고 사용자 인터페이스 등을 지원하기 위한 시스템 재개발의 필요성이 빈번해지고 있다. 시스템 리소스의 다종성(heterogeneity)은 시스템 아키텍처를 재활용 가능한 구조로 설계하기 어렵게 만드는 요인이 되고 있다.

전통적인 정보시스템의 개발에 있어서는 그림 1(a)에서 보여지듯이 공통 기능 영역에 대한 요구가 있을 때마다 따로 직접 개발해야 함으로 인해, 생산성 및 품질저하의 문제를 지니게 되었다. 최근에는 이러한 문제를 해결하기 위해서 그림 1(b)에서 보여지듯이 데이타처리, 트랜잭션 관리, 프로세스 관리, UI 개발 등을 자동화, 안정화할 수 있도록 지원하는 데이타베이스, 어플리케이션 서버, 워크플로시스템[6]과 GUI 서버 등의 다중계층의 개발 환경으로 변화되어가고 있다.

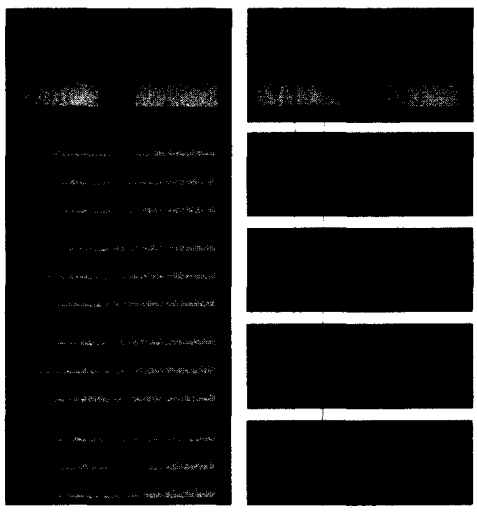


그림 1 미들웨어 또는 프레임워크에 기반한 어플리케이션 아키텍처로의 변화

그러나 이러한 다계층 환경, 즉, 역할의 분리가 이루어진 미들웨어 또는 프레임워크 들이 여러 계층과 여러 플랫폼상에서 구현된 어플리케이션 개발환경에서는, 개발 효율 및 관리적인 측면에서의 새로운 두 가지 문제가 다시 발생하게 된다. 그 첫째는, 각 계층에 중복된 서버 오브젝트[1]들의 관리를 위한 코드가 여러 클래스들에 산재하게 된다는 것이다. 예를 들어, 데이타베이스 테이블의 메타정보는 데이타베이스 서버내의 DDL(Data Definition Language), 어플리케이션 서버에서의 IDL (Interface Definition Language), 또는 UI 모듈 내에서의 정의 등으로 각각 중복 관리되는 것을 예로 들 수 있다. 그림 2는 데이타베이스 테이블의 처리에 있어서 중복되는 처리루틴과 메타정보 들의 예를 보여주고 있다. 둘째, 이러한 메타정보들을 다중의 플랫폼 상에서 상호운용하기 위해서는 상이한 데이타의 구조 및 사용언어에 기인한 문제가 발생하게 된다는 것이다[7-9].

따라서, 다종의 리소스를 수용할 수 있는 유연한 구조와 다중 플랫폼(다중 계층을 포함한)에서 공유될 수 있는 컴포넌트 구조에 대한 지원은 최근 기업 어플리케이션의 아키텍처 기반에 있어서 필수적인 부분이라고 할 수 있다.

본 논문에서 제시하는 객체지향프레임워크 SOAF (Simple Object Application Framework)은 *Adaptive Object Model*과 *XML-SOAP 인코딩 룰*을 활용하여 이러한 다종 리소스 및 다중 플랫폼의 지원에 대한 요구를 반영토록 하였다.

2.2 객체지향프레임워크

객체지향프레임워크(Object-Oriented Application Frameworks)는 어플리케이션영역의 하부 기능이나 반복코드, 전체 골격 코드들을 핫-스팟(프레임워크의 고정부위)이라는 형태로 하부에 분리시키고, 이들을 확장 및 교환하는 컴포넌트(프레임워크의 가변부위)들을 제공하여 대상 어플리케이션을 완성하도록 지원한다. 프레임워크는 Reactor[10]와 Observer[11] 같은 패턴을 적용하여, 제어권의 반전(콜백) 위주의 실행구조를 통해 실행시에 제어권을 주도한다. 이는 어플리케이션의 실행 흐름에 대한 제어권을 프레임워크로 위임하므로써, 핫-스팟의 미리 안정화된 컴포넌트 협동관계를 통하여 어플리케이션을 동작하게 한다. 이러한 설계 패러다임은 소프트웨어 개발자로 하여금 추상적 비즈니스 로직만을

1) 본 논문에서 시스템 리소스를 객체적으로 표현하여 서버 오브젝라고 칭한다.

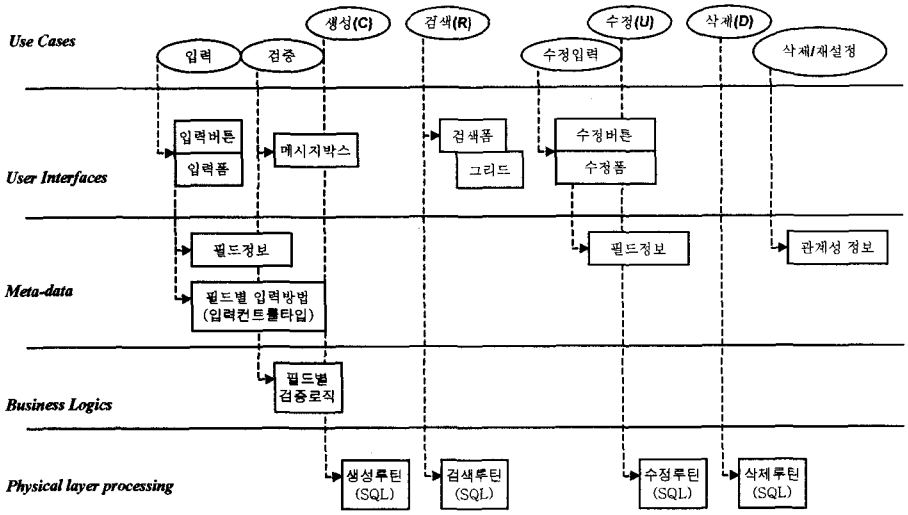


그림 2 데이타베이스 처리에서 중복관리되는 메타정보와 로직의 예

어플리케이션 컴포넌트에 구현하도록 유도하여 설계와 서비스의 질에 대한 짐을 덜게 한다[1,12].

SOAF은 기업정보시스템에 대한 분석, 설계, 구현 및 테스트 전반에 걸쳐 객체기술의 적극적 활용 및 재사용의 극대화를 통해서, 생산성 향상, 개발 품질의 고급화, 안정화 및 균일화 등을 추구하고 있다.

2.3 Adaptive Object Model

객체지향프레임워크는 시스템 구현에 있어서 재사용을 극대화하는 방법이지만, 고정된 객체모델을 기반으로 하기 때문에 새로운 요구사항을 반영하기 위한 클래스의 추가가 쉽지 않다[12]. 단적인 예로, 새로운 형식의 시스템 리소스에 대해 이를 클래스로 추상화하여 프레임워크에 추가하기가 어렵다. 최근 고정된 클래스 구조에 기반하지 않고 실행 중에 동적으로 객체구조를 읽어들여 프레임워크의 일부 혹은 전체를 구성하는 Adaptive Object Model(이하 AOM)이 제시되었다. 즉, AOM은 포괄적인(generic) 객체구조를 기반으로 실행 중에 클래스, 속성, 관계성, 행동 등 클래스의 묘사정보(메타데이타)를 이끌어낸 후 인터프리트 함으로써, 새로운 요구사항을 재컴파일 과정 없이 즉각적으로 반영하는 특성을 가진다. 이러한 적응적인 특성은 프레임워크 내에 새로운 클래스의 추가와 변경을 쉽게 하며, 묘사정보를 참조한 여러가지 부가기능(예, UI)들의 개발을 자동화할 수 있는 기반이 된다. 연구된 구체적 모델로는 Adaptive Object-Models[13], Active Object-Models [14]이 있으며, 유사한 접근방법으로 *리플렉션*[7,15], 그리고 OMG의 *Meta Object Facility*[16] 등이 있다.

그림 3은 AOM을 구현하는 방법으로 제시된 *TypeSquare* 패턴[13]을 보여주고 있다. 이 패턴은 *TypeObject, Property, Strategy, Composite* 패턴 등[11]의 조합으로 새로운 속성(Property)과 동작에 관한 규칙(Rule)을 객체인스턴스로 관리함으로써, 객체모델 자체를 유연하고 포괄적으로 표현할 수 있도록 설계되었다.

SOAF은 AOM의 접근방법과 확장된 TypeSquare 패턴을 통해 여러 종류의 시스템 리소스들을 포괄적으로 수용할 수 있는 객체구조를 정의함으로써, 프레임워크 객체모델의 적응성과 개발 자동화 영역을 증대시켰다.

2.4 XML-SOAP 인코딩 규칙

SOAP은 비중앙집중식 및 분산환경에서의 단순하고 경량인 XML기반 정보 교환 프로토콜이다. SOAP은 어

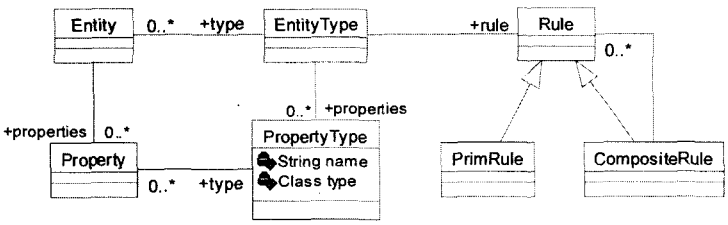


그림 3 TypeSquare 패턴[13]

플리케이션 간에 메시지를 표현하고 처리하기 위한 데이타타입들의 인스턴스를 어떻게 표현하는가를 정의하며, 권고안으로 HTTP/POST를 이용한 원격 프로시저 호출 방법을 정의하고 있다. SOAP은 다중 플랫폼간 합의된 기술인 인터넷 기술을 사용하고 소형 시스템에서도 구현 가능한 텍스트 기반이기 때문에, 플랫폼에 비종속적인 프로토콜로 주목받고 있다[17]. 특히, 포함된 정의인 *SOAP 인코딩 규칙*에서는 다중의 플랫폼과 다종 언어의 오브젝트를 지원하는 직렬화 메커니즘을 정의한다.

SOAF은 프레임워크내의 컴포넌트와 메타데이타의 상호운용성을 보장하기 위한 방법으로 XML-SOAP을 직렬화 메커니즘으로 채용하여 다중 플랫폼 환경에서의 컴포넌트 기반 개발을 가능케 한다.

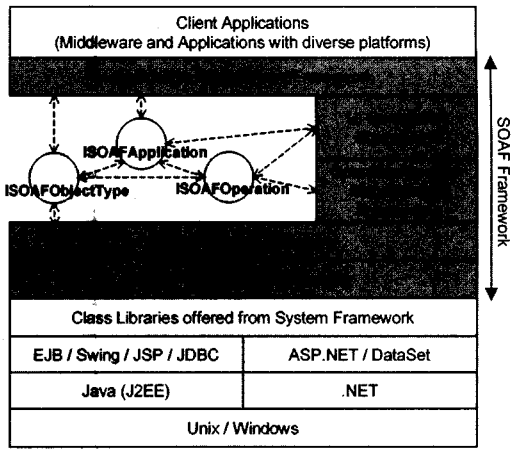


그림 4 SOAF 아키텍처

3. SOAF(Simple Object Application Framework)

AOM과 XML-SOAP인코딩 규칙을 활용한 객체지향 프레임워크 SOAF(Simple Object Application Framework)은 기능재활용성, 분산된 메타정보관리, 다종 시스템 리소스 및 다중 플랫폼의 지원 등을 해결하기 위해, 기업용 어플리케이션의 설계와 구현과정에 가이드라인이 되는 인터페이스와 고정 및 추상클래스(concrete/abstract classes)를 제공한다. 본 장에서는 객체지향프레임워크 SOAF의 아키텍처 및 주요 특징에 대해 기술한다.

3.1 SOAF의 아키텍처

SOAF은 정보시스템에서 요구되는 시스템 리소스들의 생명주기 상에 요구되는 기능들을 모아 처리할 수 있도록 프레임워크 내의 포괄적인 객체 구조와 추상적 프로그래밍 모델을 지원한다. 또한 다중 플랫폼에서의 프레임워크의 기능 접근을 위한 XML-SOAP기반의 컴포넌트 보존 방법을 정의한다. 즉, 다종의 시스템 리소스들을 일관적이고 플랫폼 독립적으로 처리하여, 어플리케이션 개발을 자동화하고 컴포넌트화 할 수 있는 기반을 제공한다. 이를 위해 SOAF은 그림 4에 표시된 5가지 서브 프레임워크와 3가지의 주요 컴포넌트 인터페이스를 정의한다. 클라이언트 어플리케이션은 각 플랫폼에 맞는 SOAF프레임워크를 하부에 가짐으로써, 상이한 플랫폼 상에서 구현되더라도 XML-SOAP 인코딩 룰에 의해 표준적으로 생성되는 컴포넌트들을 공유할 수 있게 된다.

1) **Server Object Framework**은 서버 오브젝트에 대한 메타 정보와 하부 동작 메커니즘과의 연동을 포괄적(generic)으로 저장하고 구현하는 객체구조를 제공한다. 이는 다종의 서버 오브젝트들(예를 들어, 데이타베이스 오브젝트, 메일링 서버 세션, 메시지 큐 등)이 SOAF의 서비스들을 일관적으로 제공 받도록 한다. SOAF은 서버 오브젝트들과의 표준 인터페이스로서 **ISOAFObjectType**을 정의한다.

2) **Operation/Session Management Framework**는 비즈니스 로직과 서버리소스를 관리하는 역할을 한다. 이를 통해 비즈니스 로직을 통합적이고 일관적으로 관리 할 수 있게 하고, 성능이 높고 안정적인 서버의 동작을 보장 한다. 또한 DB 커넥션, 서버 오브젝트 등의 서버 리소스에 대한 접근 및 관리 기능을 지원한다. SOAF의 표준 비즈니스 로직 컴포넌트 간의 인터페이스를 위해 **ISOAFOperation**이 정의되었다.

3) **UI Generation Framework**은 서버 오브젝트의 처리에 대한 기본적인 기능이 구현되어진 UI 컴포넌트들을 제공한다. 제공되는 UI 컴포넌트들은 이미 표준화된 정책에 따라 상호 연동 할 수 있도록 구현되어 있으므로, 서로간의 적절한 조합만으로 UI를 완성할 수 있도록 지원한다.

4) **Application Management Framework**는 클라이언트 어플리케이션 표준 인터페이스를 통해 어플리케이션들을 통합 관리하는 컨테이너 기능을 제공한다. 이를 통해 개발자 또는 사용자에게 일관된 어플리케이션 개발 및 사용 방법을 제공한다. SOAF 표준 어플리케이션과의 인터페이스를 위해 ISOAFApplication이 정의되었다.

5) **Component Framework**은 분산환경 하에서 SOAF 컴포넌트의 메타정보를 플랫폼에 종속되지 않도록 접근하기 위한 2단계 컴포넌트 보존 방법을 정의한다. 또한 컴포넌트를 빠르고 확장성 있게 개발할 수 있도록 지원하며, 다중 시스템에서 UI지원과 같은 어플리케이션 개발의 자동화를 배가시키는 기반이 된다.

3.2 Server Object Framework(SOF)

기업 어플리케이션 환경에서 사용되는 시스템 리소스

는 1) 데이타베이스, XML Repository, FTP, File System 등과 같은 '*데이타 저장소*'와 2) 메일링서버, 메시지 큐잉과 같은 '*서비스*', 그리고 3) 외부 어플리케이션, 기존 시스템과 같은 '*시스템*' 등이 있다. Server Object Framework (SOF)는 이러한 다종의 서버 오브젝트를 추상화하기 위한 포괄적인 객체구조 (인터페이스) 를 정의하여 이들이 서브 프레임워크들의 서비스를 통합적으로 지원 받을 수 있도록 한다. 이러한 포괄적 객체구조를 지원하기 위해 TypeSquare 패턴[13]의 확장모델이 활용되었다.

3.2.1 SOF의 객체구조

그림 5는 SOF의 참여 클래스와 각각의 역할을 보여준다. 서버 오브젝트인 ISOAFObject의 묘사적인 정보를 표현할 수 있는 메타데이타 및 오퍼레이션을 각각 ISOAFObjectType와 IFieldDescriptor에 정의하고 있다.

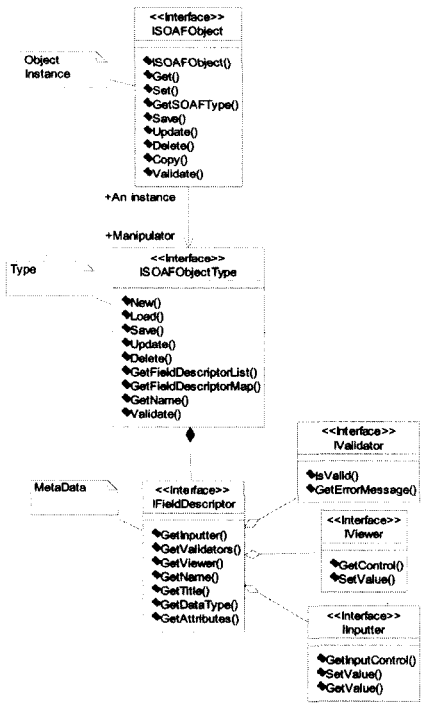


그림 5 Server Object Framework의 구조

1) Interface ISOAFObjectType: 서버 오브젝트의 형식과 행동을 일반화하고 구체화 하기 위한 인터페이스이다. 서버 오브젝트의 인스턴스(ISOAFObject)를 생성하고 처리할 때의 모든 하부 작업을 한다. 오퍼레이션은 오브젝트의 생명주기 동안에 발생할 수 있는 기본 기능인 생성, 읽기, 저장, 수정, 삭제 등과 개체 내의 메타정보(IFieldDescriptor)에 접근하기 위한 메서드 등으로 정의된다.

2) Interface ISOAFObject: 서버 오브젝트의 인스턴스 인터페이스이다. ISOAFObject의 생성, 저장, 삭제 등 인스턴스 처리에 대한 오퍼레이션은 ISOAFObjectType에 위임된다. 이는 Type object 패턴과 Factory 패턴[11]이 적용된 것으로 생성된 ISOAFObject들에 대한 추후 통제권을 확보하고 행위를 일원화하여 처리하도록 보장한다. 요소 값 (Field value)들의 키 값과 내용 값을 지정하고 얻어 낼 수 있는 접근자 오퍼레이션을 제공한다.

3) Interface IFieldDescriptor: 서버 오브젝트를 구성하는 필드에 대한 묘사정보를 정의하는 인터페이스이다. ID, 데이타형, 캡션 등의 정적 속성, 입력방법(IInputter), 검증방법(IValidator), 표시방법(IIViewer)에 대한 컴포넌트 등의 동적 속성, 그리고 확장 속성을 담고 있다. IFieldDescriptor는 이들 속성을 통해 자동 검증과 오류처리의 적절한 처리와 같은 메타데이타에 기반한 어플리케이션 개발의 자동화 영역을 넓힌다. 그림 6은 SOAFObject의 필드 값을 설정하려고 할 때 자동적으로 지정값에 대한 검증이 이루어 지는 과정을 예를 들어 보여주고 있다. SOAFObject는 IOAFObjectType과 IFieldDescriptor를 경유하여 IValidator를 참조할 수 있어, 이를 통해 얻어낸 validator로 입력된 값을 검증한다.

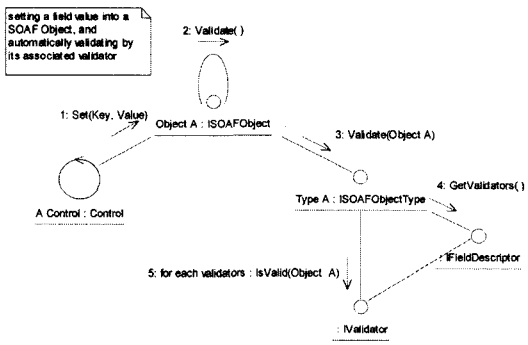


그림 6 ISOAFObject의 필드 값 설정 시에 자동적으로 행해지는 검증과정

3.2.2 SOF의 추상적 프로그래밍 스타일 지원

다종의 서버 오브젝트는 앞에서 설명한 포괄적 객체구조를 기반으로 일관적인 리소스 형태로 바인딩 될 수 있다. 예를 들어, 대표적인 서버 오브젝트인 데이타베이스 오브젝트에 대한 바인딩은 Table을 ISOAFObjectType을 구체화한 'Table'이라는 클래스명으로 구현하고, Record를 ISOAFObject를 구체화한 'Record'라는 클래스명으로 구현할 수 있다. 다른 예로, Email 서

비스에 대한 바인딩으로 메일을 보내고 얻어오는 메일 서버 기능은 ISOAFObjectType을 구체화한 'Email-Server'라는 클래스명으로 구현할 수 있고, 각각의 메일은 ISOAFObject를 구체화 한 'Email'이라는 클래스명으로 구현할 수 있다. 리스트 1은 다종의 서버 오브젝트들의 바인딩을 바탕으로 일관적인 서버 오브젝트의 처리와 통합을 가능케 하는 추상적인 프로그래밍 스타일을 보여주고 있다.

리스트 1. 다종의 리소스를 통합하는 SOAF의 추상적 프로그래밍 스타일

```
// 사용할 리소스 타입 클래스를 얻어온다.
Table addressbookTable = session.getSOAFObjectType("AddressBookTable");
EmailServer mailServer = session.getSOAFObjectType("EmailServer");

// 주소록 데이타베이스에서 주소와 메일을 보낼 주소를 얻어온다.
Record address = addressbookTable.load("name='장진영'");

// 메일을 보내기 위한 인자를 설정한다.
Email newEnvelope = mailServer.New();

newEnvelope.set("to", address.get("email"));
newEnvelope.set("from", "kinam@abc.com");
newEnvelope.set("contents", "abcde....");

// 메일을 보낸다.
newEnvelope.save();
```

3.3 Operation/Session Management Framework (OSMF)

그림 7은 Operation/Session Management Framework(OSMF)의 참여 클래스의 정의와 협력 클래스들과의 관계를 보여주고 있다. OSMF는 Command 패턴[11]을 채용하였다. 참여 클래스는, SOAF에서 정의된 리소스들(서버 오브젝트 및 로직)을 처리하는 표준 로직 인터페이스의 정의인 *ISOAFOperation*과 이의 실행 관리와 리소스 접근을 위한 세션의 정의인 ISOAF-Session이다. 이들은 3.5절에서 설명될 *ISOAFPortal* 및 *ISOAFApplication* 클래스와 사용자 인터페이스를 위해 협동한다.

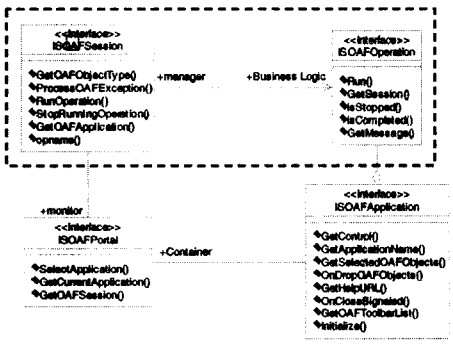


그림 7 Operation/Session Management Framework의 구조

3.3.1 OSMF의 객체구조

1) Interface ISOAFSession: ISOAFOperation의 동작 시 실제 관리를 위한 구현이 이루어 진다. 또한 서버 리소스, 인증 및 접근제어 등을 관리하는 역할도 담당한다.

2) Interface ISOAFOperation: SOAF의 표준 로직을 위한 인터페이스이다. 로직 수행 시 사용자 제어, 안정성, 퍼포먼스 등의 측면에서 필요한 메서드들의 선언으로 이루어진다. 이를 통해 비즈니스 로직을 통일된 형태로 컴포넌트화 할 수 있으며, 프레임워크 전반을 통한 안정되고 풍부한 기능지원이 가능해 진다. 그림 8은 이러한 예로서, SOAFOperation의 동작 중에 사용자 제어 및 모니터링 과정을 보여 주고 있다.

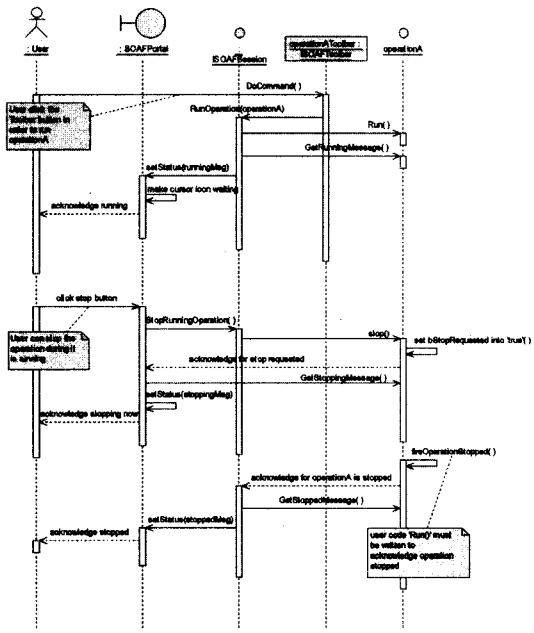
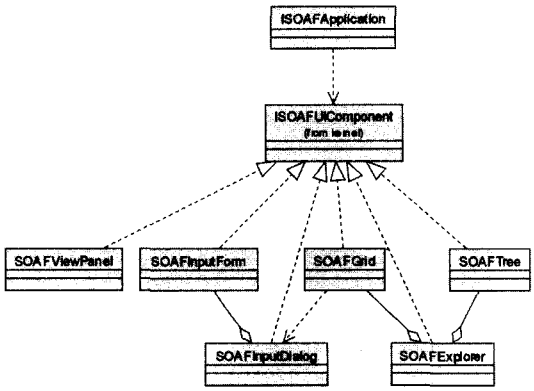


그림 8 SOAFOperation의 호출과정

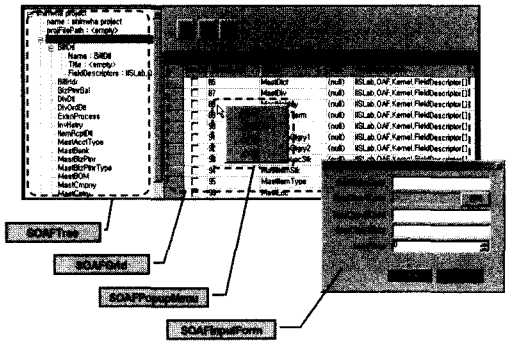
3.4 UI Generation Framework(UGF)

UI Generation Framework(UGF)은 SOAFObject를 처리하기위한 사용자와의 인터페이스가 필요할 때, 해당 UI를 자동으로 생성하도록 지원하고 있다. 그림 9(a)는 SOAF에서 지원하는 여러 UI 컴포넌트 간의 관계성을 보여주고 있으며, 그림 9(b)는 자동 생성된 UI의 한 예를 통해 각 컴포넌트에 대응되는 일부를 보여 주고 있다.

UI의 자동생성과정은 SOAF 서버 오브젝트의 각 필드별 메타정보들 (IInputter/IViewer)에 기초하여 이루



(a) 클래스 Hierarchy



(b) 자동생성된 UI의 예

그림 9 SOAF UI 컴포넌트들

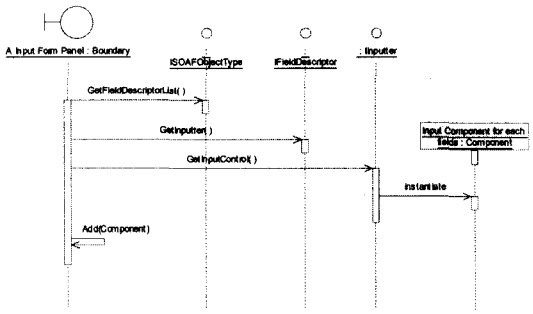


그림 10 InputForm(입력폼)의 생성과정

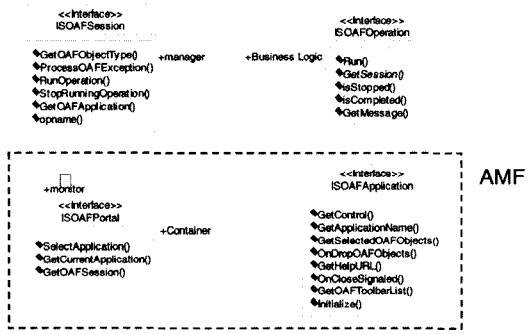


그림 11 Application Management Framework의 구조

어진다. 예를 들면 각 필드의 데이타형에 맞는 값을 입력 받을 컨트롤들로 구성된 폼의 생성, 특정 이벤트에 대한 다이얼로그 생성, 에러 메시징 등은 미리 구현된 UI 컴포넌트들에 의해 지원된다. 그림 10은 데이타입력을 위한 가장 기본적인 UI 컴포넌트인 입력폼 (InputForm)의 생성 과정을 보여준다.

메타데이타에 기반한 자동 UI 생성 메커니즘은 새로이 요구되는 환경의 UI를 추후에 계속적으로 지원해갈 수 있는 바탕이 된다. 예를 들어 웹이나 모바일 등과 같은 새로운 UI환경에 대한 요구가 발행할 시에 서버오브젝트에 대한 메타데이타를 참조하여 ISOAFUIComponent를 상속하는 해당 컴포넌트들을 추가함으로써 처리할 수 있다.

3.5 Application Management Framework(AMF)

Application Management Framework(AMF)는 최종 사용자와 SOAF 리소스들간의 인터페이스 역할을 한다. SOAF은 포괄적인 표준 리소스와 로직 인터페이스를 정의하여 통합된 상위 UI개발과 클라이언트 어플리케이션의 재사용을 지향한다. 즉, ISOAFObject를 데이타 단위로, ISOAFOperation을 이들을 처리하기 위한 로직으로, 그리고 UI를 생성해주는 수단으로 제공된 UI 컴포넌트들을 사용해, 미리 관계설정을 이루어 놓은 클라이언트 어플리케이션을 지원한다. 이는 어플리케이션 개발자간 인터페이스를 원활히 하고 사용자에게 일관된 어플리케이션 사용 방법을 제공하는 효과가 있다. 그림 11은 이러한 기능을 담당하는 AMF의 구성 클래스인 ISOAFPortal과 ISOAFApplication을 보여주고 있으며, 3.3 절에서 설명한 OSMF의 ISOAFSession과 ISOAFOperation과 협력하고 있다.

3.5.1 AMF의 객체구조

1) Interface ISOAFApplication: SOAF 클라이언트 어플리케이션의 표준 인터페이스의 정의이다.

2) Interface ISOAFPortal: 여러 개의 SOAFApplication을 관리하는 컨테이너 역할을 구현한다.

3.6 SOAF Component Framework(SCF)

SOAF은 내부에서 활용되는 컴포넌트를 다중 시스템에서 교환 및 운용하고 툴의 지원에 의해 컴포넌트를 생성하기 위해, 2단계에 걸친 컴포넌트 보존 및 재생 방법을 정의한다. SOAF은 컴포넌트에 포함되는 메타정보

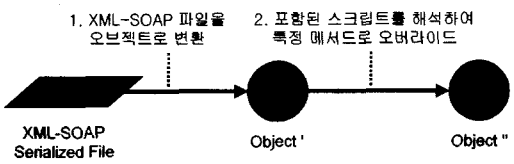


그림 12 SOAF의 2단계 컴포넌트 재생 메커니즘

와 속성을 SOAP 인코딩 룰[18]에 기반 하여 보존 시키며, 행위부분은 텍스트 기반의 스크립트 언어를 사용하여 해당 XML에 포함시킨다. 그림 12는 보존된 컴포넌트를 재생하는 과정을 보여주고 있으며, 보존과정은 역순으로 진행된다.

3.6.1 XML-SOAP을 통한 컴포넌트 보존

오브젝트를 직렬화 하기 위해서 메모리 내에 존재하는 오브젝트를 리플렉션 API[2] 등을 통해 필드구조 및 클래스 정보 등을 얻어내어 SOAP 인코딩 룰에 따라 직렬화한 후 파일로 저장한다. 저장된 XML 파일을 하나의 컴포넌트로 다룰 수 있다.

3.6.2 스크립트 언어를 통한 객체 행위(로직) 보존

컴포넌트의 메타정보는 SOAP 인코딩 룰에 의해 보존될 수 있는 데이타 성격이지만, 컴포넌트의 특정 행위를 다중 플랫폼에서 교환할 수 있기 위해서는 행위자체가 특정언어에 종속되지 않는 텍스트기반의 형태로 전달되어야 한다. 이를 위해 자바스크립트와 같이 표준화된 스크립트 언어를 사용했다. 이를 위해 SOAF Component Framework은 특정 메서드에 대한 요청에 대해 해당 스크립트를 인터프리트한 후, 그 결과를 돌려주는 프록시 클래스를 리턴한다. 이 프록시 클래스는 내부적으로 자바의 *Bean Scripting Framework*[19]나 닷넷의 *Common Language Infrastructure*[20] 등을 사용해서 스크립트를 수행시켜 결과를 제공할 수 있다.

이 두가지 컴포넌트 보존 메커니즘을 통해서, SOAF의 모든 컴포넌트는 상이한 플랫폼과 프로그래밍 언어에서 구현된 SOAF-호환 프레임워크에서 동작할 수 있게 된다. 또한 IDE툴을 통해 컴포넌트를 개발하고 프레임워크에 탑재시킬 수 있는 기반도 제공한다. 이를 통해, 성능개선을 위해 C++로 개발된 서버에서 생성된 서버 오브젝트의 FieldDescriptor를 자바로 개발된 클라이언트에서 접근하여 사용할 수 있게 하는 등의 장점도 낳게 된다.

4. 적용사례: 적응형 워크플로관리시스템 uEngine

워크플로관리시스템(Workflow Management System; WfMS)[6]은 사람과 IT리소스 간의 협업으로 이루어지는 비즈니스 프로세스를 정의하고 실행하는 자동화 환경이다. WfMS는 프로세스와 이의 구성요소들인 단위업무로 나뉘는 2-단계 어플리케이션 개발방법[9]을 통해, 프로세스를 중심으로한 대규모 기업 어플리케이션의 효율적인 개발과 유지보수를 유도한다. *uEngine* [21-23]은 새로운 기술변화에 대응하는 액티비티 유형의 추가에 대한 적응성과 다중 플랫폼의 지원에 있어서 유연성이 부족한 기존 워크플로시스템의 문제점을 해결하기 위해, SOAF을 기반으로 웹서비스 기술을 활용하여 개발된 적응형 워크플로관리시스템이다. 그림 13은 서버측에서 수행되는 *Workflow Enactment Server*는 J2EE/Linux환경으로, 사용자에게 여러가지 클라이언트 툴을 제공하는 *Worklist Handler Server*는 .NET/Windows로, 그리고 관리자용 *Process Designer*는 Java/Windows환경에서 예시적으로 구현된 uEngine의 다중 플랫폼 구조를 보여주고 있다. uEngine은 그림 13에서 나타난 것과 같이 WfMS의 각 구성요소에 포함된 SOAF서브프레임워크들의 계층을 활용하여 다종리소스와 다중플랫폼의 지원에 있어서의 적응성을 높이고 있다.

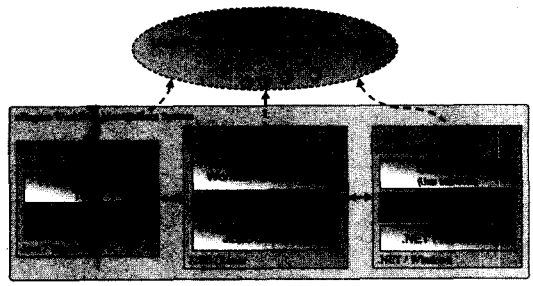
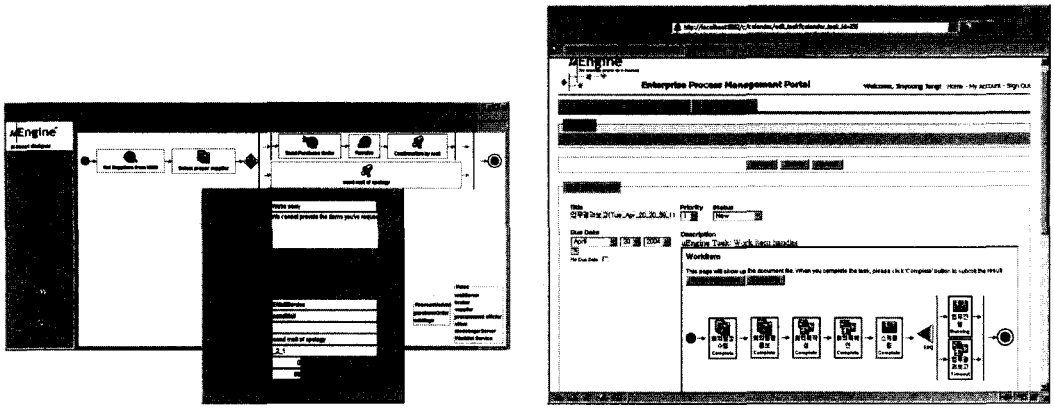


그림 13 SOAF을 기반으로 구축된 적응형 워크플로관리시스템 uEngine의 구조

그림 13에서와 같이 시스템의 구성요소들이 분리된 환경에 존재할 경우에는, 각 플랫폼에서 분산적으로 처리하게 되는 서버 리소스들이 존재하게 된다. 예를 들어, 프로세스 흐름 간 데이타 전달을 위한 *프로세스 변수*[6]의 입력을 위한 UI의 개발은 Worklist Handler에, 그것의 보존에 대한 처리는 Enactment Server로 분산된다. uEngine은 이와 같은 다중 플랫폼 상에 분산된 리소스 관리의 문제를 해결하기 위해 액티비티 유형과 프로세스 변수 타입들을 SOAF의 CF를 통해 SOAF 컴포넌트들로 생성한 후 네트워크상에서 공유한다. 이로써 각 리소스에 대한 묘사정보를 분산된 워크플로 구성요소에서 중복 관리하지 않고 중앙의 컴포넌트의 변화가 모든 구성요소에 일관적으로 반영되도록 한다. 이를 통해 각 구성요소들이 상이한 플랫폼 상에 존재하면서

2) 객체의 속성 및 메서드 정보를 동작 중에 얻어내고 호출 할 수 있도록 지원된 API



(a) (b)

그림 14 SOAF 컴포넌트를 공유하는 uEngine Process Designer(a)와 Worklist Handler의 화면(b)

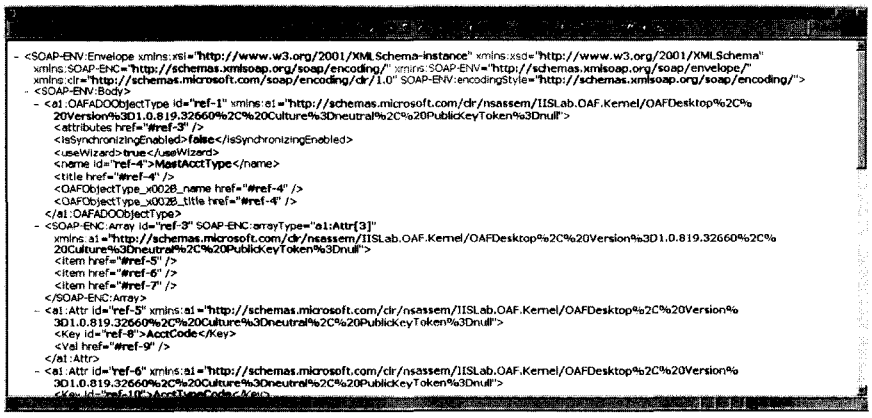


그림 15 SOAF 컴포넌트 프레임워크를 통해 생성된 XML-SOAP형식의 SOAFObjectType 컴포넌트

도 각 플랫폼의 특징적인 장점[3]을 활용할 수 있게 한다. 그림 14는 자바와 닷넷환경에서 실제 구현되어 중앙의 서버 리소스에 대한 메타정보를 공유하는 Process Designer와 Worklist Handler의 화면을 보여주고 있다. 이때 UGF는 각각의 UI 제공을 자동화하는 역할을 수행한다. 그리고 Workflow Enactment Server에 구현된 SOF 서브프레임워크는 포괄적 객체구조를 통해 새로운 기능을 반영하는 액티비티 유형의 추가를 쉽게 함으로써 고정된 객체모델에 기반한 기존 워크플로의 단점을 해소하고 있다.

자바버전의 SOAF구현에는 데이타베이스 접근과 XML-SOAP 인코딩을 위해서 JDBC와 오픈소스 라이브러리인 아파치 액시스[24]를 활용하고 있으며, 닷넷버전의 SOAF 구현에는 UI제공과 XML-SOAP 인코딩을 위해 닷넷의 *Web Forms, Windows Forms, SoapFormatter*[25] 등의 라이브러리가 활용하였다. 그림 15는 두 버전의 CF에 의해 상호운용될 수 있도록 XML-SOAP 인코딩 룰에 의해 직렬화된 SOAF 컴포넌트의 예를 보여주고 있다.

5. 결론 및 향후 연구 계획

본 논문에서 제시한 SOAF(Simple Object Application Framework)은 다종의 시스템 리소스를 지원해야 함과 동시에 다중 플랫폼 및 분산환경의 계층을 지닌 시스템 환경에서 전반적으로 적용 가능한 컴포넌트 기반 개발 환경이다. SOAF은 다음과 같은 2가지 특성을 가지고 있다.

첫째, 다종의 시스템 리소스를 수용할 수 있는 객체구조를 지원한다. Adaptive Object Model[13,14]에 기반

3) 마이크로소프트의 *닷넷 프레임워크*[25]와 자바의 *J2EE*[26]는 XML, 웹서비스, 데이타베이스 등의 e-비즈니스 및 기업어플리케이션을 지하기들을 통합제공하는 경쟁적인 양대 플랫폼이다. 윈도 운영체제에 기반한 마이크로소프트의 닷넷은 다양한 클라이언트 어플리케이션들의 지원이용이하며, 닷넷보다 역사가 오래된 J2EE는 서버측 어플리케이션 영역에서 적용된 사례가 많다.

한 유연한 객체구조를 지닌 Server Object Framework와 Operation/Session Management Framework의 포괄적인(generic) 시스템 리소스 및 업무로직의 바인딩은, SOAF 프레임워크 전반에서 고수준으로 추상화 된 통합 개발을 가능케 하고 있다.

둘째, 다중 플랫폼에서 컴포넌트 기반 개발을 할 수 있도록 지원한다. XML-SOAP 인코딩 룰과 표준 스크립트 언어에 기반한 Component Framework에 의한 컴포넌트 보존 메커니즘은 상호운용성이 있는 컴포넌트를 생성하게 함으로써, 닷넷 및 자바, C++ 등 하부 개발언어 에 종속되지 않고 공유될 수 있는 플랫폼 독립적 컴포넌트 기반 개발을 가능케 한다.

본 논문에서는 SOAF 프레임워크의 핵심 인터페이스와 프로토콜에 대한 내용만을 기술하였다. SOAF 자체만으로도 어플리케이션 구축에 있어서 많은 효과를 제공하지만, 보다 생산적이고 객체지향개념에 근접한 개발을 유도하기 위해서는 마법사형태의 어플리케이션 생성툴도 유용하게 활용될 수 있다[27,28]. 향후 SOAF에 기반한 다양한 e-비즈니스 솔루션의 개발, 그리고 최근에 주목받고 있는 웹서비스 아키텍처와의 통합 방안에 대한 연구도 가치가 있으리라 본다.

참 고 문 헌

[1] Johnson, R., "Frameworks = (components + patters)," Communications of the ACM, vol. 40, no. 10, pp. 39-42, October 1997.

[2] JCorporate Ltd., "Expresso Project," http://www.jcorporate.com/, October 2003.

[3] Monday, P., M. Dangler, and J. Carey, SanFrancisco component framework: an introduction, Addison-Wesley, 2000.

[4] McGovern, J., S. Tyagi, M. Stevens, and S. Mathew, Java Web Services Architecture, Morgan Kaufmann, 2003.

[5] Maamar, Z., Q.Z. Sheng, and B. Benatallah. "On composite web services provisioning in an environment of fixed and mobile computing resources," Information and Technology Management, http://www.cse.unsw.edu.au/qsheng/papers/ITM-03.pdf, 2003.

[6] Workflow Management Coalition, The Workflow Reference Model: Document Number TC00-1003, 1995, available at http://www.wfmc.org.

[7] Blair, G.S., Coulson, G., Robin, P., and Papathomas M., "An Architecture for Next Generation Middleware," Proc. of the Int. Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Springer, 1998.

[8] Fugetta, A., Picco, G. P., and Vigna, G., "Understanding Code Mobility," IEEE Transactions on Software Engineering, vol. 24, no. 5, 1998.

[9] Ledoux, T. and Bouraqadi-Saadani, N. "Adaptability in Mobile Agent Systems using Reflection," RM'2000, Workshop on Reflective Middleware, http://www.comp.lancs.ac.uk/computing/rm2000/, 2000.

[10] Schmidt, D.C., "Reactor: An Object behavioral pattern for concurrent event demultiplexing and event handler dispatching," In Pattern Languages of Program Design (Coplien, J.O. and Schmidt, D.C., eds.), pp. 529-545, Addison-Wesley, 1995.

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

[12] Fayad, M.E. and D. Schmidt, "Object-Oriented Application Frameworks," Communications of the ACM, vol. 40, No.10, pp. 32-38, October 1997.

[13] Yoder, J. W. and R. Johnson, "The Adaptive Object Model Architectural Style," http://www.adaptiveobjectmodel.com/WICSA3/ArchitectureOfAOMsWICSA3.pdf, 2002.

[14] Foote, B. and Yoder, J. W., "Metadata and Active Object Models," Technical Report wucs-98-25, Dept. of Computer Science, Washington University, http://jerry.cs.uiuc.edu/~plop/plop98, October 1998.

[15] Smith, B.C., "Procedural Reflection in Programming Languages," PhD Thesis, MIT, MIT Laboratory of Computer Science Technical Report 272, Cambridge, Mass.

[16] Object Management Group. Meta Object Facility (MOF) specification. Technical Report MOF V1.3 RTF, Object Management Group, September 1999.

[17] Seely, S., SOAP: Cross Platform Web Service Development Using XML, Prentice Hall, 2001.

[18] W3C, Simple Object Access Protocol, http://www.w3.org/TR/2000/NOTE-SOAP-20000508, May 2000.

[19] IBM Corporation, Bean Scripting Framework, http://oss.software.ibm.com/developerworks/projects/bsf, October 2003.

[20] Microsoft Corporation, "Common Language Infrastructure", http://msdn.microsoft.com/netframework/using/understanding/cli/default.aspx, May 2004.

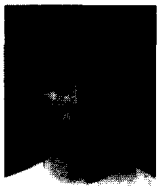
[21] Jang, J., Y. Choi and J. L. Zhao, "An Extensible Workflow Management Architecture with Web Services," International Journal of Web Services Research, vol. 1, no. 2, pp. 1-15, 2004.

[22] Jang, J., Y. Choi and J. L. Zhao, "Adaptive Workflow Management with Open Kernel Framework Based on Web Services," Proc. of the 1st Int. Conf. on Web Services, pp. 124-130, 2003.

[23] uEngine.org, uEngine wfms project, http://www.uengine.org, 2004.

[24] Apache Software Foundation. Apache Axis project, http://ws.apache.org/axis/index.html, May 2004.

[25] Microsoft Corporation, "Microsoft .NET Framework Developer Center," http://msdn.microsoft.com/netframework/, May 2004.
[26] Sun Microsystems, Inc., "JavaTM 2 Platform Enterprise Edition Specification, v1.4," April 2003, http://java.sun.com
[27] 인제대학교 BPM 실험실, SOAF 프로젝트, http://bpm.inje.ac.kr/soaf/index.html.
[28] 장진영, 이성용, 최용선, "e-Business Application 개발을 위한 객체지향 프레임워크와 컴포넌트 생성 도구", 한국전자거래학회 종합학술대회 논문집, pp. 595-602, 2002.



장 진 영

BPM 솔루션 업체인 (주)핸디소프트에 연구원으로 근무했으며 현재는 (주)현세시스템의 소프트웨어 아키텍트로 근무중이다. 인제대학교 대학원 광대역정보통신공학과 공학석사를 취득했다. 관심분야는 객체지향분석과 프레임워크, 워크플로와 웹서비스를 기반한 기업간 거래, 오픈소스 비즈니스 모델 등

최 용 선

인제대학교 시스템경영공학과에 부교수로 재직중이다. 서울대학교에서 학사, 그리고 한국과학기술원에서 석사, 박사 학위를 각각 산업공학 전공으로 취득하였다. 주요 관심분야는 Workflow & Business Process Management, Web Services, e-Business, Component Based Development, Object-oriented System Analysis & Design 등