상호운용성 평가 정보 관리를 위한 J2EE 기반의 LISI 저장소 개발 프레임워크

(LISI Repository Development Framework based on J2EE for Interoperability Assessment Information Management)

조 정 희 † 정 병 훈 †† 염 근 혁 †††
(Jung Hee Cho) (Byeonghoon Cheong) (Keunhyuk Yeom)

요 약 최근 정보체계 개발에 있어서 체계간 상호운용성의 중요성으로 인해 상호운용 능력을 평가하는 정보체계 상호운용성 수준(LISI)이 정의되어 있다. 이 때, 상호운용 평가 자료의 저장 및 관리를 효과적으로 지원하기 위한 평가 정보 저장소의 구축이 필수적으로 요구된다.

평가 정보의 저장 및 관리를 위한 LISI 저장소는 다양한 플랫폼에서 개발된 평가체계에 독립적이면서 표준화된 방법을 통해 그 기능을 제공할 수 있어야 한다. LISI 저장소의 비즈니스 로직들은 현재의 분산 환경에 맞도록 분산 컴포넌트로 개발되어져야 하며 표준화된 LISI 데이타의 표현과 함께 LISI 저장소의 특성을 잘 반영할 수 있도록 기존의 소프트웨어 설계 방법을 보다 구체화할 필요가 있다. 뿐만 아니라 LISI 저장소의 구현에 필요한 다양한 구현 기술들을 인식하고 하나의 LISI 저장소의 구현에 통합 활용할 수 있도록 하는 방안이 필요하다. 본 논문에서는 상호운용성 평가 정보를 관리하는 LISI 저장소의 설계 및 구현에 있어서 LISI 저장소의 특성을 3가지 관점에서 고려하여 LISI 저장소에 구체화한 설계 방안과, 다양한 LISI 저장소 구현 기술을 하나의 LISI 저장소 구현을 위해 상호 유기적으로 통합 활용할 수 있는 구현 방안을 LISI 저장소 개발 프레임워크로서 제시한다. 이는 실제 설계자 및 개발자들에게 LISI 저장소의 구축 시 고려되어야 할 요소와 적용 방법들에 대한 구체적인 가이드라인을 제시해줌으로써 보다 쉽고 효과적인 LISI 저장소의 구축을 돕는다.

키워드 : 정보체계 상호운용성 수준, 평가 정보 저장소, 분산 컴포넌트, LISI 저장소 개발 프레임워크

Abstract Recently, Levels of Information Systems Interoperability (LISI) that are used for assessing the information systems interoperability are defined according to the importance of interoperability among information systems. At that time, it is necessary to the construction of an assessment information repository to store and retrieve the interoperability assessment data efficiently.

An LISI repository constructed for storing and retrieving an assessment information should support assessment systems that are developed in various platforms and provide its functions by a standardized way. Business logics of LISI repository should be developed as distributed components, and we need to modify existing software design methods to show characteristics of LISI repository in addition to the standardized LISI data expression. Also a concrete method to recognize various implementation technologies and to apply them to implementation of LISI repository is needed. This paper provides a design method for developing an LISI repository that is considering the characteristics of LISI repository by three concerns, and an implementation scheme that can apply to the integration of the various repository implementation technologies systematically, as the LISI repository development framework for constructing the LISI repository. These help developers to construct the LISI repository effectively by providing concrete guidelines for considerations and appliable methods to construct the LISI repository.

Key words : Levels of Information Systems Interoperability (LISI), Assessment Information Repository, Distributed Components, LISI Repository Development Framework

· 본 연구는 국방과학연구소 위탁과제 연구비와 부산대학교 기성회재원 연구비 지원으로 수행되었음
† 비 회 원 : 국방과학연구소 연구원
etrange7@dreamwiz.com
†† 비 회 원 : LG정보통신
geonmo@hanmail.net
††† 종신회원 : 부산대학교 컴퓨터공학과
yeom@pusan.ac.kr
논문접수 : 2003년 6월 30일
심사완료 : 2004년 9월 6일

1. 서 론

최근 정보체계 개발에 있어서 체계간의 상호운용성 및 통합 실행 환경의 중요성이 널리 인식되고 있다. 이로 말미암아 정보체계의 통합 상호운용의 소요를 결정하고 이를 충족시키기 위한 능력을 평가하는 정보체계 상호운용성 수준(LISI: Levels of Information System Interoperability)이 정의되어 있다[1,2]. LISI를 바탕으로 상호운용성 수준을 평가하는 평가체계는 테스트 시나리오, 테스트 데이타 및 다양한 평가 대상체계의 개발 문서 등을 종합적으로 활용하여 평가를 수행한다. 뿐만 아니라, 체계 개발자를 포함한 많은 평가체계 이용자들은 각자 정보체계의 상호운용 요구사항을 정의할 때, 기존 체계에 대한 상호운용 규격 등의 자료 참조를 필요로 한다. 따라서 상호운용 평가에 있어 효과적인 자료의 저장 및 관리를 지원하기 위한 LISI 저장소의 구축이 필수적으로 요구된다.

LISI 저장소는 다양한 플랫폼에서 구축된 평가체계의 서브시스템에 의해 공유되기 위해서 평가체계에 독립적이어야 하며 표준화된 방법을 통해 그 기능을 제공할 수 있어야 한다. LISI 저장소 내부의 저장, 검색 등의 비즈니스 로직은 현재의 분산 환경에 맞도록 분산 컴포넌트로서 개발되어야 한다. 또한 상호운용성 평가 정보인 LISI 데이타는 특정 플랫폼으로부터 중립성을 지켜주는 XML(eXtensible Markup Language)[3] 형태로서 정의하여 사용하는 것이 필요하다. 이와 같은 특징을 갖는 LISI 저장소를 효과적으로 설계하기 위해서 컴포넌트 기반의 소프트웨어 설계 방법을 이용해야 하지만, 기존의 소프트웨어 설계 방법은 그 활용이 광범위하고 일반적이다. 따라서 실질적인 설계를 위해서는 LISI 저장소의 외부적 환경을 비롯한 기능적/비기능적 특성과 구현 기술들을 다양한 관점에서 반영하여 추가할 필요가 있다. 뿐만 아니라 LISI 저장소의 구현을 위해서 이용되어지는 다양한 기술들을 통합하여 활용할 수 있는 구현 방안이 필요하다.

본 논문에서는 상호운용성 평가 정보를 관리하는 LISI 저장소의 효과적 구축을 위한 설계 및 구현 프레임워크를 제시한다. 이를 위하여 LISI 저장소의 특성을 논리적, 물리적, 데이타의 3가지 관점으로 나누어 모델화 함으로써 LISI 저장소의 설계를 위한 구체적인 설계 방안을 제시한다. 또한 XML에 의한 LISI 데이타의 표현 기술과 컴포넌트 모델을 통한 분산 지원 기술 및 웹 서비스 기술 등의 LISI 저장소 구현 기술들에 대한 역할과 기술들 간의 상호관계를 정의함으로써 다양한 LISI 저장소 구현 기술들을 상호 유기적으로 통합 활용할 수 있는 구현 방안을 제시한다. 컴포넌트 모델은 다양한 플랫폼 지원과 상태 관리의 이점을 가지고 있는 J2EE(Java 2 Enterprise Edition)[4]의 EJB(Enterprise JavaBeans)[5]를 기반으로 한다.

2. 관련 연구

2.1 정보체계 상호운용성 수준(LISI)

LISI는 정보체계 상호운용성을 정의하고 측정하며 평가하기 위한 절차이다. 이는 정보체계의 능력 평가를 위한 성숙도 모델과 평가 프로세스를 제시하고 더 높은 수준의 능력과 상호운용성을 위한 해결책 및 대안을 제시하는데 그 목적이 있다[2].

평가를 수행하는 상호운용성 평가체계는 그림 1에서와 같이 LISI 수집체계(Inspector)와 LISI 보고체계(LISI Reports)의 두 서브시스템으로 구성되어 있다. LISI 수집체계는 평가체계 이용자로부터 체계에 관한 정보를 수집하여 상호운용성 프로파일을 생성하며, LISI 보고체계는 상호운용성 프로파일을 통해 대상체계 및 체계간의 상호운용성 수준을 비교 분석하여 매트릭스와 인터페이스 다이어그램 등의 다양한 형태로 보여준다. 평가체계는 평가 프로세스를 수행하는데 있어 효과적인 LISI 데이타의 저장 및 관리를 위해 LISI 저장소와 상호 작용한다. 이때 LISI 저장소에서 관리되어야 할 정보는 평가 프로세스 단계에 따라 상호운용성 수준 평가의 기준을 제시하는 기초 정보와 평가 대상체계의 정보 수집에 관련된 질의 정보, 그리고 이를 통해 생성되는 산출 정보의 3가지 유형으로 분류할 수 있다. 표 1은 LISI 저장소에서 관리되어야할 유형별 LISI 데이타를 설명한다.

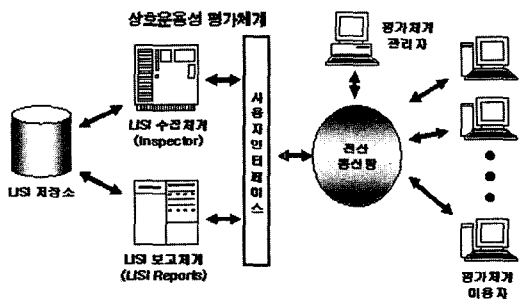

그림 1 상호운용성 평가체계 구성도

2.2 재사용 저장소 및 컴포넌트 저장소

재사용 저장소는 재사용 가능한 소프트웨어 Asset을 정의하고 저장하며 관리하기 위한 메커니즘으로 라이브러리와 같은 개념이다[6]. 좋은 저장소는 인터넷을 통해 사용자의 접근과 운용이 쉬워야하고 재사용 Asset에 대

표 1 LISI 데이타 집합

관리 데이타 유형	LISI 데이타	내 용
기초 정보	성숙도 모델	체계간의 상호 작용에 대한 복잡도를 5단계의 상호운용성 수준으로 정의한 모델
	참조 모델	성숙도 모델의 각 수준에 대해 이해하기 쉬우면서 통합된 속성인 PAID로 묘사한 모델
	능력 모델	PAID 속성으로 기술된 각 수준을 획득하기 위해 필요한 능력 집합을 정의한 모델
	구현 옵션 테이블	능력 모델에서 식별된 각 능력을 구현하기 위해 체계 개발자가 이용할 수 있는 모든 가능한 구현 옵션을 포함하고 있는 테이블
질의 정보	상호운용 질의서	평가 대상체계의 능력에 대한 이용 가능한 구현 옵션사항과 체계에 구현된 서비스를 포함한 구조화된 질문들
	질의응답	상호운용 질의서에 대한 체계 개발자의 응답 내용
산출 정보	상호운용성 프로파일	상호운용 질의서를 통해 얻은 평가 대상체계의 데이타를 능력 모델 템플릿으로 매핑한 것으로서 평가 대상체계의 PAID 능력에 대한 구현 선택을 기록한 데이타

한 정형적인 표현을 제공해야 한다. 많은 조직에서 그들의 소프트웨어 재사용 요구에 맞는 도메인 저장소들을 개발하였으며 Software Asset Library Management System(SALMS)[8], Comprehensive Approach to Reusable Defense Software(CARDS), Asset Source for Software Engineering Technology(ASSET), Defense Software Repository System(DSRS) 등이 있다[7,8].

최근의 많은 소프트웨어들이 컴포넌트 기반으로 개발됨에 따라 적절한 컴포넌트의 획득 및 관리가 소프트웨어 개발에서의 중요한 부분이 되었으며 재사용 저장소는 컴포넌트 저장소로 발전하게 되었다. 컴포넌트 저장소는 재사용 가능한 컴포넌트를 위치시키고 검색하며 관리를 지원하는 라이브러리 시스템으로서 소프트웨어의 신뢰성 및 생산성을 향상시키고 중복 개발을 최소화한다. 컴포넌트 저장소가 가지는 중요한 능력으로 다음과 같은 내용이 포함된다[7].

- 컴포넌트의 브라우징과 검색을 위한 사용자 중심의 GUI를 지원하는 자동화된 라이브러리 시스템
- 표준적인 컴포넌트 프레임워크 (예. 목적, 기능성 서술, 인증 수준, 환경 제약 등)
- 각 도메인에 대한 효과적인 분류 방안
- 완전한 시스템 및 컴포넌트 문서

현재 상용적인 컴포넌트 저장소로는 ComponentSource[9], 컴포넌트 뱅크[10] 등이 있다. 이 중 가장 규모가 큰 ComponentSource는 현재 9000여개의 등록된 프로덕트들을 컴포넌트, 도구, Add-in, 개발 플랫폼으로 나누어 하위에 90여 가지의 카탈로그와 함께, 각 카탈로그에서 도메인에 해당하는 세분화된 카테고리를 제공한다. 따라서 카테고리 별로 프로덕트를 검색할 수 있으며 키워드 기반 검색도 가능하다. 그러나 카테고리는 그 분류의 기준이 없고 단일 분류로만 구성되어 있어 체계적이지 못하다.

2.3 컴포넌트 기반 소프트웨어 설계 방법

최근 소프트웨어는 CCM(CORBA Component Model)[11], COM(Component Object Model)[12], EJB등의 컴포넌트 모델을 이용하여 컴포넌트 기반으로 개발되고 있다. 컴포넌트 기반 소프트웨어 개발(CBSD: Component Based Software Development)은 이미 존재하는 소프트웨어 컴포넌트를 조립함으로써 시스템을 개발하는 방법이다[13]. 대표적인 방법으로 Catalysis, Unified Process, CBD96 등이 있다.

Catalysis는 UML(Unified Modeling Language)[14]을 사용하여 객체와 컴포넌트 기반 소프트웨어 개발을 지원하는 방법론으로 도메인/비즈니스 레벨, 컴포넌트 명세 레벨, 내부 설계 레벨의 세 레벨로 구성된다[15]. 또 다른 컴포넌트 기반 개발 방법론으로 Unified Process는 역시 UML을 이용하며 도입(Inception) 단계, 정제(Eleboration) 단계, 구축(Construction) 단계, 전이(Transition) 단계의 네 단계로 구성되어 있고, 각 단계 내부는 요구사항, 분석, 설계, 구현, 테스트로 된 다섯 개의 핵심 워크플로우로 구성된다[16]. CBD96은 Sterling에서 제시한 컴포넌트 기반 개발 방법으로 Catalysis를 단순화하고 체계화하여 확장한 방법이며[17,18], 프로세스는 요구사항 정의, 분석, 행위 명세, 아키텍처의 네 단계로 이루어진다.

이러한 방법론들은 컴포넌트 기반의 소프트웨어 개발을 위한 절차 및 작업을 지원하고 있으나, 그 적용이 일반적이므로 실제 특정 소프트웨어를 개발하기 위해서는 개발하고자 하는 소프트웨어의 특성을 반영하여 보다 구체적이고 다양한 관점에서의 설계 지침들이 필요하다.

2.4 LISI 저장소를 위한 소프트웨어 구현 기술

LISI 저장소의 구현을 위해 필요한 기술로는 LISI 저장소에 저장할 단위 데이타의 표현 문제를 해결할 정보

표현 기술과 분산 환경에서 컴포넌트 단위로 구성될 LISI 저장소가 영향을 받게 될 각종 비기능적 특성 문제를 해결할 분산 환경 지원 기술, 그리고 LISI 저장소가 개발언어, 운영체제, 미들웨어 등 특정 플랫폼으로부터 독립적인 시스템으로 구축되도록 도와주는 웹 서비스 기술이 있다.

정보 표현의 대표적인 기술인 XML은 웹 상에서 데이타 교환을 위한 표준으로서 부각되고 있는 텍스트 기반의 마크업(markup) 언어이다. XML은 여러 태그를 이용하여 데이타를 식별한다는 점에서는 HTML과 유사하지만, HTML이 데이타의 디스플레이에 중점을 두는데 비해 XML은 데이타 자체의 식별에 중점을 둔다. 분산 환경을 지원하기 기술로서 컴포넌트 모델은 설계된 개개의 컴포넌트가 어떻게 만들어질 것인가를 정의하고 만들어진 각 컴포넌트들이 어떻게 상호 작용할 것인가를 정의하기 위해서 필요하다. 대표적인 컴포넌트 모델인 EJB는 기업 환경의 어플리케이션을 개발하고 배치하기 위한 소프트웨어 컴포넌트 모델 또는 서버 측 컴포넌트 아키텍처로서, 개발자가 짧은 시간 내에 신뢰할 수 있고 확장 가능하며 이식성 높은 안전한 컴포넌트를 만들 수 있도록 해준다. XML 웹 서비스[19-21]는 WWW 인프라를 확장하여 소프트웨어를 다른 소프트웨어 어플리케이션으로 연결하는 수단을 제공한다. 어플리케이션은 웹 서비스가 어떻게 구현되는지 알 필요 없이 HTTP, XML, 그리고 SOAP(Simple Object Access Protocol)과 같은 편재한 웹 프로토콜과 데이타 포맷을 통하여 웹 서비스에 접근하게 된다. 웹 서비스는 사용자와 대화할 수 있는 클라이언트 어플리케이션을 만들기 위한 인터페이스를 세부적으로 설명하는 WSDL(Web Service Description Language) 문서를 제공하며, 잠재적인 사용자가 웹 서비스를 쉽게 찾을 수 있도록 UDDI (Universal Description, Discovery, and Integration)에 등록된다.

그러나 앞서 언급한 기술들은 현재 각기 독립적인 이용 방안만이 존재하므로 LISI 저장소의 구축을 위해서는 각 개별 기술들을 LISI 저장소를 위한 하나의 통합 기술로서 활용할 수 있는 구현 지침이 필요하다.

3. LISI 저장소 개발 프레임워크

본 장에서는 LISI 저장소의 효과적인 구축을 위한 설계 방안 및 구현 방안을 통해 LISI 저장소 개발 프레임워크를 제시한다.

3.1 LISI 저장소의 설계 방안

LISI 저장소는 다양한 플랫폼에서 구축된 평가체계에 독립적이며 어플리케이션 수준의 인터페이스로서 LISI 데이타의 저장, 검색 및 관리 서비스를 제공해야 한다. 이러한 LISI 저장소를 설계하기 위해서는 다양한 측면에서의 LISI 저장소의 특성이 반영되어야 한다. 즉, LISI 저장소의 논리적인 구성 측면과 실제 구현 기술에 특화된 측면, 그리고 특별히 LISI 저장소 내부 LISI 데이타의 구조 측면에서 LISI 저장소의 특성을 반영한 설계 지침이 필요하다. 따라서 본 논문에서는 LISI 저장소의 특성을 그림 2와 같이 논리적, 물리적, 데이타의 3가지 관점으로 나누어 모델화함으로써 LISI 저장소의 설계를 위한 구체적인 방안을 제시한다. LISI 저장소의 논리적 모델은 LISI 저장소의 기능을 논리적 관점으로 분석하여 구조화하며, 물리적 모델은 논리적 모델에 존재하는 논리적 구성 요소들이 LISI 저장소를 구현하게 되는 기술 요소로 어떻게 연결될 수 있는지의 관점으로 구현 기술에 특화된 기능 및 관계를 정의한다. 그리고 데이타 모델은 LISI 저장소 내 데이타의 동적 흐름과 LISI 데이타의 정적 관계를 정의한다.

그림 2 LISI 저장소의 설계 방안

3.1.1 논리적 모델

LISI 저장소의 논리적 모델은 LISI 저장소의 구조와 기능을 나타낸 모델로서, 기술적인 자세한 사항을 추상화시키면서 구현 기술에 독립적으로 기능적인 컴포넌트들과 그들간의 상호 관계를 표현한다. 이러한 논리적 모델은 LISI 저장소 설계자에게 LISI 저장소의 기본적인 구성 요소와 각 요소의 책임 및 역할, 그리고 요소사이의 관계에 관한 정보를 제공함으로써 LISI 저장소의 요구사항을 추출하고 분석하는 가이드를 제시한다. 또한 논리적 모델은 LISI 저장소를 이용하는 외부 사용자의 요청을 만족시키는 비즈니스 로직 측면에서 LISI 저장소의 기능성 및 비기능성들을 논리적 구성 요소로 매핑시켜 정의한다.

그림 3에서 보는 바와 같이 LISI 저장소는 3 계층 클라이언트-서버 아키텍처를 바탕으로 하여 논리적으로 크게 인터페이스 층(LISI Repository Interface)과 비즈

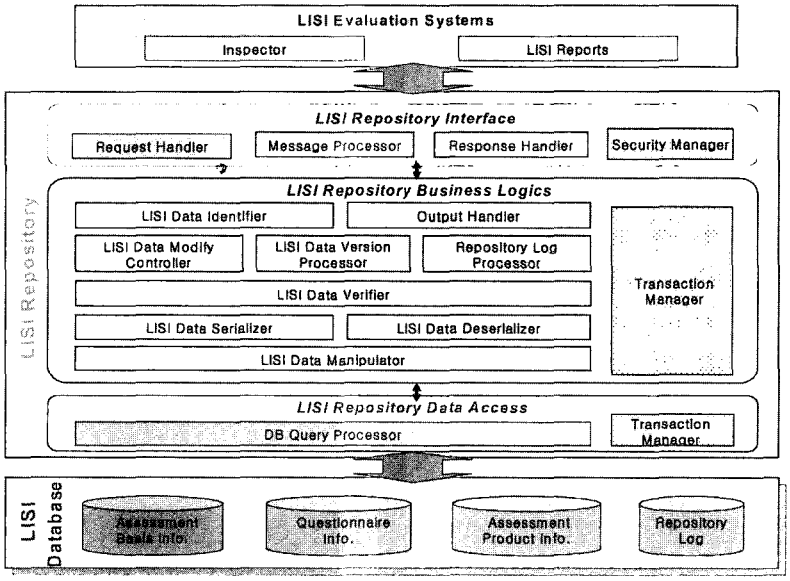

그림 3 LISI 저장소의 논리적 모델

니스 로직 층(LISI Repository Business Logics), 그리고 데이타 접근 층(LISI Repository Data Access)으로 구조화 되어있다. 이와 같이 구조화함으로써 LISI 데이타의 검증, 변환 등의 중요한 비즈니스 로직을 외부와의 상호작용 및 데이타베이스 접근으로부터 분리할 수 있다. LISI 저장소의 외부에는 LISI 저장소를 이용하는 평가체계 시스템(LISI Evaluation Systems)인 Inspector와 LISI Reports가 존재하며, LISI 저장소에서 관리되는 정보의 유형별 데이타베이스(LISI Database)가 존재한다.

• 인터페이스 층

LISI 저장소 인터페이스 층은 LISI 저장소 외부의 평가체계 서브시스템인 Inspector및 LISI Reports와 상호작용하는 층으로서, LISI 저장소의 가장 바깥에서 평가체계로부터 LISI 저장소의 저장, 검색, 관리의 서비스 요청을 받아들인다(Request Handler, Message Processor). 그리고 받아들인 외부의 요청을 처리하는 LISI 저장소 비즈니스 층으로 이 요청을 전달하고 처리된 요청 결과에 대한 응답을 평가체계로 반환하는 역할(Response Handler, Message Processor)을 담당한다. 또한 인증된 평가체계만 LISI 저장소로 접근 가능하게 하는 비기능성을 수행한다(Security Manager).

• 비즈니스 로직 층

LISI 저장소의 비즈니스 로직 층은 앞서 인터페이스 층으로부터 받아들여진 서비스 요청을 처리하는 층으로서, 실제 LISI 저장소의 주요 기능을 처리하는 역할을 담당한다. 즉, 인터페이스 층으로부터 LISI 데이타의 저장, 검색 및 수정 등의 서비스 요청에 대한 각 요청을 식별하고(LISI Data Identifier), LISI 데이타를 관리한다(LISI Data Version Processor). 또한, LISI 데이타의 검증 및 LISI 저장소 내의 추상화된 데이타 형태(ADM: Abstract Data Model)로의 변환을 수행하고(LISI Data Verifier, LISI Data Serializer, LISI Data Deserializer, LISI Data Manipulator) 해당 LISI 데이타에 대해 유형별로 데이타베이스에 저장하거나 데이타베이스로부터의 검색을 위한 처리를 하여 그 결과를 반환한다(Output Handler, Repository Log Processor). 뿐만 아니라, 동시에 요청된 서비스들을 관리하고 네트워크 문제 및 오류들을 처리하는 비기능성도 수행한다(Transaction Manager).

• 데이타 접근 층 및 LISI 데이타베이스

LISI 저장소 데이타 접근 층은 LISI 데이타를 유형별로 추상화한 LISI 데이타베이스로의 접근 로직을 제공하는 역할을 한다. 즉, 비즈니스 로직 층에서 각 요청을 처리하기 위해서는 필수적으로 데이타베이스로의 접근이 필요하며(DB Query Processor), 이는 요청된 서비스나 LISI 데이타의 종류에 독립적으로 존재 가능하므로 비즈니스 로직으로부터 분리시킴으로써 독립성을 유지시킨다. 그리고 각 LISI 데이타의 유형별로 저장 및 관리되는 3개의 데이타베이스(Assessment Basis Info., Questionnaire Info. and Assessment Product Info.)와 로그정보를 저장하는 데이타베이스(Repository Log)가 존재한다.

3.1.2 물리적 모델

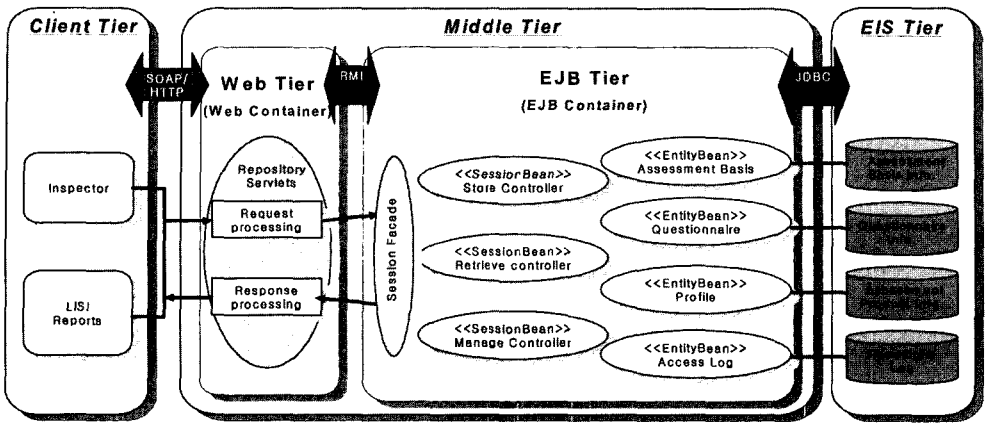

그림 4 LISI 저장소의 물리적 모델

LISI 저장소의 물리적 모델은 LISI 저장소의 구축에 있어서 물리적 환경의 설계 제약 조건을 논리적 모델에 적용한 형태로서, LISI 저장소 구현 기술 연구를 통해 정의된 구현 프레임워크에 특화하여 LISI 저장소의 기능과 관계를 표현한다. 이러한 물리적 모델은 LISI 저장소 설계자에게 LISI 저장소의 구현 기술이 반영된 설계 정보를 제공함으로써 LISI 저장소의 논리적 요소들을 특정 구현 기술에 맞게 구체화하여 설계할 수 있는 가이드를 제시한다. LISI 저장소의 물리적 모델은 논리적 모델에서의 최소 단위의 기능성을 J2EE 아키텍처 기반의 EJB 컴포넌트 모델 및 웹 서비스의 구성 요소로 매핑함으로써 정의한다.

그림 4에서 보는 바와 같이 LISI 저장소는 J2EE 아키텍처를 바탕으로 하여 LISI 저장소를 이용하는 외부 평가체계 시스템인 클라이언트 계층(Client Tier), LISI 저장소에 해당하는 중간 계층(Middle Tier)으로서 웹 계층(Web Tier)과 EJB 계층(EJB Tier), 그리고 LISI 저장소를 이용하는 LISI 데이타의 RDBMS에 해당하는 정보 시스템 계층(EIS Tier)으로 구조화된다. 클라이언트 계층과 EIS 계층은 LISI 저장소와 상호 작용하는 계층이며 실제 LISI 저장소에 해당하는 계층은 중간 계층의 웹 계층과 EJB 계층이다.

• 웹 계층

LISI 저장소의 웹 계층은 웹 컨테이너로서 동작하는데 이는 Inspector와 LISI Reports로부터 SOAP 메시지 형태의 서비스 요청을 받아들이고 이를 적절하게 처리하는 비즈니스 로직으로 전달하는 역할을 한다. 물리적 모델의 웹 계층은 논리적 모델의 인터페이스 층에 해당하는 요소들이 구현 기술 요소로 매핑된 것이다. 웹 컨테이너에는 LISI 저장소 서블릿(Servlet)이 존재하여 서비스 요청과 응답을 처리한다(Request Processing, Response Processing). 즉, 웹 계층에서는 클라이언트 계층의 Inspector와 LISI Reports로부터 들어오는 LISI 저장소 서비스에 대한 SOAP 메시지 요청을 핸들링하고 실제 이를 처리하는 비즈니스 로직이 존재하는 EJB 계층으로 Session Facade 패턴을 통해 전달한다. 이와 함께 SOAP 헤더를 통해 인증된 평가체계가 LISI 저장소를 사용할 수 있도록 하는 인증 허가 등의 보안 서비스를 호출한다.

• EJB 계층

EJB 계층은 EJB 컨테이너로서 동작하는데 실제 LISI 저장소의 주요 기능인 검색, 저장, 관리 로직을 서비스 단위로서 처리하는 하는 역할을 한다. 물리적 모델의 EJB 계층은 논리적 모델의 비즈니스 로직 층과 데이타 접근 층에 속하는 요소들이 EJB의 구현 요소로 매핑된 것이다. EJB 계층은 웹 계층으로부터 Inspector와 LISI Reports의 요청 서비스를 전달받으며(Session Facade), 각 서비스 기능과 요청된 LISI 데이타의 유형별로 EJB의 구성 요소인 세션 빈(Store Controller, Retrieve Controller, Manage Controller)과 엔티티 빈(Assessment Basis, Questionnaire, Profile, Access Log)을 통해 요청을 처리한다. 그리고 저장 또는 검색될 LISI 데이타를 유형별로 데이타베이스에 저장하거나 검색한다. LISI 데이타 유형별 실제 데이타베이스는 JDBC(Java Database Connectivity)를 통하여 접근 가능하며 이는 EJB의 컨테이너를 사용하여 관리를 용이하게 한다. 또한 비기능성과 관련하여 식별된 보안과 트랜잭션 요소는 J2EE 프레임워크에서의 API(Application Programming Interface)를 이용하여 구현한다. 즉, 인증된 평가체계의 LISI 저장소 접근만을 허용하는 보안 요소는 JAAS(Java Authentication and Authorization Service)를, 동시에 요청된 서비스를 처리하는

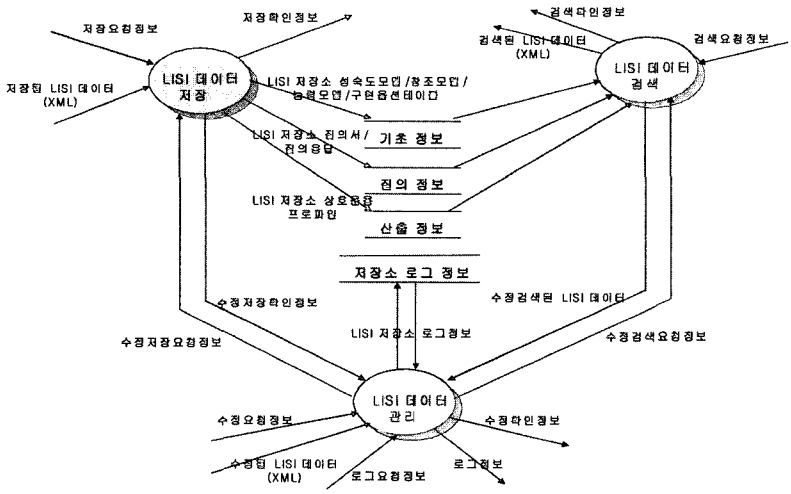

그림 5 LISI 저장소의 데이타 흐름

트랜잭션 요소는 JTA(Java Transaction API)를 이용하여 실현한다.

3.1.3 데이타 모델

LISI 저장소의 데이타 모델은 LISI 저장소와 관련한 데이타 전반을 나타낸 모델로서, LISI 저장소의 논리적 기능성 및 구현 기술과는 별도로 LISI 저장소가 다루고 있는 LISI 데이타들의 흐름 및 상관 관계를 표현한다. 이러한 데이타 모델은 LISI 저장소 설계자에게 논리적 모델과 함께 동적인 측면에서 LISI 저장소를 분석하는 가이드를 제시하며, 또한 물리적 모델과 함께 EJB의 엔티티 빈으로 LISI 데이타를 구조화할 수 있는 가이드를 제시한다. LISI 저장소의 데이타 모델은 LISI 저장소의 요구사항과 LISI 저장소의 논리적 모델을 바탕으로 LISI 저장소의 기능을 만족시키는 프로세스를 세부 프로세스의 집합으로 분할을 통해 다차원적으로 접근하여 정의한다. 또한 평가체계 내에서 많은 연관성을 가지고 정의되어 있는 LISI 데이타들 사이의 관계를 정적인 측면에서 구조화하여 정의한다. 이러한 두 가지 측면에서의 데이타 모델은 다음과 같다.

• LISI 데이타의 동적 흐름

LISI 저장소 내의 각 프로세스를 따라 데이타가 흐르면서 변환되는 모습을 DFD(Data Flow Diagram)로서 표현하면 그림 5와 같다. 일반적인 DFD는 프로세스를 강조하지만 LISI 저장소에 대한 데이타 모델로서의 DFD는 논리적 모델의 기능성을 바탕으로 하는 프로세스들의 입출력 데이타에 초점을 둔다.

그림 5는 LISI 저장소의 기본 기능인 저장, 검색, 관리 기능 전체에 대한 데이타의 흐름을 표현한다. 외부로부터 입력되는 서비스 요청 정보와 저장 및 수정될 XML 형태의 LISI 데이타는 각 기능에 대한 프로세스를 거쳐 LISI 저장소 내의 데이타 형태(ADM)로 변환되고 이들이 데이타 스토어에 저장되고 검색된다. 그리고 처리 결과에 대해 서비스 요청 확인 정보 및 검색된 LISI 데이타를 출력한다.

• LISI 데이타의 정적 관계

LISI 저장소에서 관리되는 LISI 데이타에서는 기초 정보인 성숙도 모델, 참조 모델, 능력 모델 및 구현 옵션 테이블과 질의 정보인 상호운용 질의서 및 질의 응답, 그리고 산출 정보인 상호운용 프로파일이 존재하며, 이들 정보는 상호 일반화 또는 구체화, 포함 관계 등의 연관성을 지니고 있다.

그림 6은 이러한 LISI 데이타들의 속성과 연관 관계로서 LISI 데이타들간의 정적 관계를 정의한다. 성숙도 모델은 LISI 내에 표현된 상호운용성 수준을 정의하는 모델로 상호운용성 정도에 따른 수준을 가지며 각 수준은 세부 속성(정보교환, 상호협력, 자료 및 응용 관계, 컴퓨팅 환경)으로 특징지어진다. 참조 모델은 성숙도 모델에서 정의한 각 수준을 4가지 속성인 절차, 응용체계, 기반구조, 그리고 데이타(PAID: Procedure, Application, Infrastructure and Data)로 세분화한다. 능력 모델은 참조 모델의 PAID 속성에 대한 능력을 포함하며 추가적으로 각 수준은 다시 부수준 ID와 PAID를 속성을 가지도록 정의된다. 능력 모델의 각 수준별 PAID 속성은 그 능력을 구현하는 여러 구현 옵션을 가지고 있으며, 이 데이타는 세부 속성(DITA 분류, Capability, Technical Profile#)을 가지도록 정의된다. 상호운용 질의서는 질의 ID와 구현 옵션으로 구성된 질의 항목을 속성으로 가지며, 앞서 언급한 구현 옵션 데이타와 포함

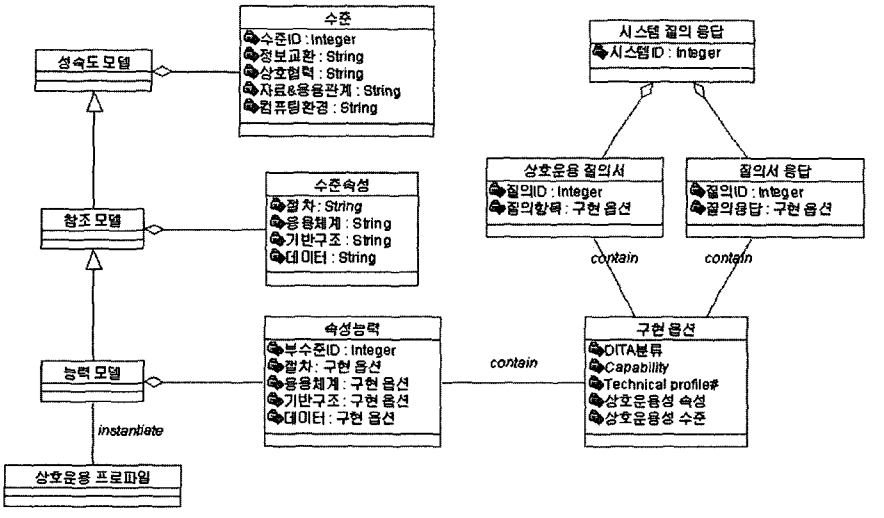

그림 6 LISI 데이타의 정적 관계

관계로 정의된다. 이와 함께 질의서 응답은 평가된 체계에 대한 상호운용 질의서의 응답 사항이며 질의서와 같은 형태이다. 이러한 상호운용 질의서와 질의서 응답으로써 하나의 평가 대상체계에 관한 시스템 질의 응답이라는 데이타를 정의한다. 마지막으로 상호운용 프로파일은 질의서를 통해 얻은 평가 대상 체계의 데이타를 능력 모델 템플릿으로 매핑한 결과이다. 따라서 이는 능력 모델 데이타와 인스턴스화(Instantiate)의 관계로 연결된다.

3.2 LISI 저장소의 구현 방안

설계된 LISI 저장소를 실제 구현하기 위해서는 다양한 구현 기술들이 필요하며 이러한 기술들은 하나로 통합되어져서 이용될 수 있어야 한다. 이를 위하여 본 논문에서는 설계 방안에 따라 설계된 LISI 저장소의 구현 시 필요한 각 기술 요소들의 역할을 정의하고 기술 요소들 간의 상호 관계를 파악함으로써 하나의 LISI 저장소 구현을 위한 통합 구현 기술 프레임워크를 제시한다.

LISI 저장소를 구현에 필요한 구현 기술은 크게 3가지로 나누어진다. 첫 번째로 LISI 저장소에 저장되는 LISI 데이타를 표현하기 위한 기술 요소로서 XML이 필요하다. 그리고 LISI 저장소 설계 방안의 물리적 모델에서 제시한 컴포넌트들은 분산 환경에 위치할 수 있는데, 이러한 분산 환경 하에서 컴포넌트들의 정확한 동작을 위한 규칙들을 표준으로 정의하고 있는 컴포넌트 모델이 또한 필요하다. LISI 저장소는 그것의 특성상 비즈니스 계층에서 관리해 주어야 할 엔티티가 다수 존재하며 엔티티들을 영구적으로 관리하는 것이 중요하므로 LISI 저장소의 컴포넌트 모델로서 J2EE 기반의 EJB를 선정하였다[26]. 마지막으로 LISI 저장소가 다른 플랫폼에서 개발된 평가체계와 상호 작용하는 연결 수단을 제공하는 웹 서비스 기술이 필요하다.

LISI 저장소에 필요한 각 구현 기술들은 설계 방안에서 제시한 설계 내용을 실현하기 위하여 기술 요소간에 상호 유기적으로 동작할 수 있어야하므로 본 논문에서는 LISI 저장소의 구현 기술들을 그림 7과 같이 하나의 통합 프레임워크로 정의한다.

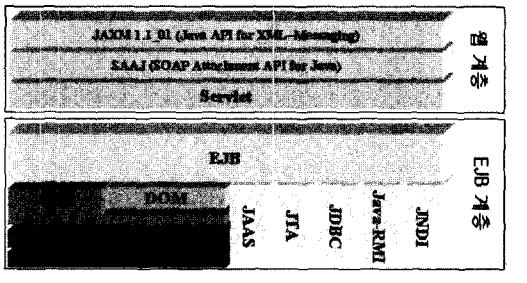

그림 7 LISI 저장소의 구현 기술 프레임워크

그림 7의 LISI 저장소의 구현 기술 프레임워크는 설계 방안의 물리적 모델에서 식별된 바와 같이 웹 서비스로서의 상호 작용을 수행하는 웹 계층과 LISI 저장소로서의 비즈니스 로직을 수행하는 EJB 계층으로 크게 나누어진다. 평가체계가 LISI 저장소의 서비스를 웹 서비스 형태로 요청하면 LISI 저장소의 웹 계층은 그러한 요청을 식별하여 EJB 계층에 다시 요청한다. EJB 계층은 LISI 저장소의 비즈니스 로직을 수행 후 그 결과를 웹 계층에 반환하며, EJB 계층으로부터 결과를 반환 받은 웹 계층은 다시 웹 서비스의 형태로써 서비스를 요청한 평가체계에 결과를 반환한다.

EJB 계층에는 LISI 데이타와 관련한 XML 기술과 J2EE의 다양한 분산 지원 기술이 존재하고 그 위에 컴포넌트를 동작시키는 EJB가 존재한다. 그리고 웹 계층은 웹 서비스의 창구 역할을 하는 서블릿과 그 위에 SOAP 메시지를 처리하는 SAAJ(SOAP Attachment API for Java)와 JAXM(Java API for XML-Messaging)으로 구성된다. 전체적으로 상위 계층의 기술은 하위 계층의 기술 지원을 통하여 완전히 실현될 수 있다.

3.2.1 EJB 계층

설계 방안에서 이미 EJB 계층의 역할을 정의하고 이 계층을 형성하는 컴포넌트를 세션 빈, 엔티티 빈의 형태로 식별하였다. 여기서 식별된 각 EJB 빈들은 XML을 통한 데이타 표현 기술 및 EJB를 포함한 여러 J2EE의 분산 지원 기술들로써 실현된다. 다음에서는 EJB 계층에 속하는 XML을 통한 데이타 표현 기술과 EJB를 포함한 분산 지원 기술들의 LISI 저장소에 대한 역할 및 기술들 간의 상호 관계를 설명한다.

• XML을 통한 데이타 표현

XML을 통한 데이타 기술들은 세부적으로 XML과 XML 스키마, SAX (Simple API for XML), 그리고 DOM(Document Object Model)로 구성된다.

- **XML** : XML은 LISI 저장소와 평가체계가 독립적으로 개발 및 상호 작용이 가능하기 위해 특정 플랫폼으로부터 중립성을 지켜주는 데이타 형태로서 LISI 데이타를 정의하고 사용할 수 있도록 한다. 따라서 평가체계와 LISI 저장소가 주고받는 상호운용성 프로파일 및 상호운용 질의서 등의 LISI 데이타를 XML형태로 정의한다. 이 때 XML 형태로 데이타를 정의하기 이전에 XML 데이타를 형성하는 원칙을 제공하는 메타데이타가 필요한데 이를 XML 위쪽에 존재하는 XML 스키마가 지원한다.
- **XML 스키마** : XML 스키마는 앞서 XML을 통해 상호운용성 프로파일 및 상호운용 질의서 등의 LISI 데이타를 표현할 때의 메타데이타를 정의할 수 있도록 한다. 따라서 LISI 데이타들 중 상호운용성의 기준을 제시하는 성숙도 모델, 참조 모델, 능력 모델, 그리고 구현 옵션들이 XML 스키마로서 정의된다. XML의 메타데이타를 구성하는 형식에는 EBNF(Extended Backus-Naur Form) 문법을 사용하는 DTD(Document Type Definition)와 XML 형식을 준수하는 XSD(XML Schema Definition)가 있는데[22], LISI 저장소에서 정의할 LISI 데이타들은 메타데이타의 조작성(manipulability)과 유효성 검증(validatability) 능력의 측면에서 XSD로서 표현한다.
- **SAX와 DOM** : DOM과 SAX는 모두 XML 형태의 데이타를 처리하는데 필수적인 기술들로서 앞서 XML과 XML 스키마를 통해 정의한 LISI 데이타를 파싱(parsing)하는 역할을 한다. 설계 방안의 물리적 모델에서 제시한 EJB 계층 내 Profile Controller, Questionnaire Controller 등의 각 엔티티 빈에서 LISI 데이타를 ADM 형태로 변환 또는 그 역 과정을 수행 시 이 DOM과 SAX를 이용하게 된다. SAX는 그 기능적 특성으로 인해 XML 데이타를 쓸 수만 있지만 DOM은 XML 데이타의 읽기와 쓰기가 동시에 가능하다. 따라서 LISI 저장소의 구현 시 DOM만을 이용해서도 LISI 저장소의 모든 XML 파싱 기능을 수행할 수 있지만 DOM만을 이용하는 것은 성능과 메모리 사용성 측면에서 좋지 않다. 이러한 이유로 SAX는 LISI 저장소에서 LISI 데이타의 저장 요청과 함께 입력된 XML 데이타의 읽는 역할을, DOM은 LISI 데이타의 검색 요청 시 ADT 형태의 LISI 데이타를 XML 형태로 구성하기 위해 쓰는 역할을 담당한다.

• EJB를 포함한 J2EE 분산 지원

J2EE 분산 지원 기술들은 세부적으로 JNDI(Java Naming and Directory Interface), Java-RMI(Remote Method Invocation), JDBC, JTA, JAAS로 구성되고 이들 위에 EJB가 존재한다.

- **JNDI** : JNDI는 EJB에서 여러 빈들의 EJBHome 객체를 찾는 메커니즘을 제공하는 기술로서[23], LISI 저장소 설계 방안의 물리적 모델에서 제시한 EJB 계층의 빈들이 다른 빈들을 찾을 수 있도록 한다. JNDI를 통하여 EJB는 장소에 대한 투명성을 얻을 수 있고 따라서 네트워크를 통해 사용되는 EJB의 여러 자원들은 장소에 독립적으로 동일한 형식을 통하여 이용된다.
- **Java-RMI** : Java-RMI는 LISI 저장소 내에서 원격 객체간에 정보 교환을 가능하게 하는 역할을 한다. EJB 및 J2EE의 하부 기술들로써 구축이 될 LISI 저장소는 그 내부의 여러 구성 요소들이 Java 개발 언어를 통해 개발되며 이러한 Java 기반의 객체 사이에 원격 호출 관계가 존재할 때 Java-RMI 기술로써 해당 호출을 구현할 수 있다. 따라서 앞서 설명한 JNDI를 통하여 서비스를 받고자 하는 EJB의 빈들을 찾아내고 Java-RMI를 통해 찾은 EJB 빈의 외부로 드러낸 메쏘드를 이용한다.
- **JDBC** : JDBC는 엔티티 빈들이 LISI 데이타를 RDBMS로 저장하거나 혹은 RDBMS로부터 검색할 때 필요한 기능을 제공하는 기술로서 대부분 설계 방안에서 정의한 엔티티 빈이 RDBMS로 접근할

때 이용된다.

- **JTA** : JTA는 LISI 저장소의 수행에 있어 데이타베이스로의 접근 외의 트랜잭션 관리가 필요한 부분들에 관해 개별적으로 여러 트랜잭션 속성을 설정함으로써 LISI 저장소의 트랜잭션 처리의 완성도를 높여주는 역할을 한다. 트랜잭션은 일반적으로 RDBMS가 수행될 때 쓰여지는데, 앞서 기술한 JDBC는 RDBMS의 트랜잭션을 자동으로 관리해 주고 있고, CMP(Container Managed Persistence)를 사용할 경우에는 EJB 컨테이너에서 트랜잭션을 관리해 주고 있다. 그러나 보다 세부적으로 트랜잭션을 사용하기 위해서는 JTA가 필요하게 되는 것이다[24].
- **JAAS** : JAAS는 LISI 데이타의 보안적 특성을 실현하기 위한 기술로서[24] 이를 이용하여 LISI 저장소가 인증된 평가체계에게만 서비스를 제공하도록 하는 인증 절차를 실현할 수 있게 된다.
- **EJB** : 마지막으로, EJB는 앞서 언급한 여러 기술들의 지원을 통해서 완성된다. 즉, LISI 저장소 설계 방안의 물리적 모델에서 식별한 EJB 빈들은 JNDI를 이용하여 다른 빈들을 찾고, 원격 객체의 경우 Java-RMI를 통해 필요한 메쏘드를 호출하게 된다. 그리고 LISI 데이타의 DBMS 접근을 위해 JDBC를 이용하며 그 외 트랜잭션 관리는 JTA를 통하여 구현한다. 또한 JAAS를 이용하여 보안을 위한 인증 절차를 구현한다.

3.2.2 웹 계층

웹 계층은 평가체계의 요청을 HTTP 형식으로 입력받아 전달된 SOAP 메시지의 처리를 수행 후 EJB 계층의 Session Facade 세션 빈을 통해 LISI 저장소의 서비스를 호출한다. 이는 크게 서블릿과 JAXM(Java API for XML-Messaging) 기술들로써 실현된다. 다음에서는 웹 계층에 속하는 이 두 가지 기술들의 LISI 저장소에 대한 역할 및 기술들 간의 상호 관계를 설명한다.

• 서블릿

서블릿은 LISI 저장소에서 웹 서비스의 창구 역할을 하며 서블릿 엔진이라고도 불리는 웹 컨테이너에 의해 상시 수행되면서 평가체계가 요청한 HTTP 형태의 서비스에 대해 적절하게 대응하여 EJB 컨테이너의 Session Facade 세션 빈을 호출한다.

• JAXM

JAXM은 SOAP 메시지를 메시징(Messaging) 방식으로 처리하는 기술이다. SOAP 메시지를 처리하는 방식은 SOAP-RPC(Remote Procedure Call)와 SOAP 메시징(Messaging)이 존재하는데[25], 평가체계가 LISI 저장소에 대해 서비스를 호출한 후 다음 수행을 신속하게 하기 위해서는 LISI 저장소와 평가체계 간의 비동기 방식을 지향해야 하며, 또한 평가체계가 제공하는 LISI 데이타는 신뢰성 있게 전달될 수 있어야 한다. 이러한 기능은 SOAP 메시징 방식을 통해 가능하며, 따라서 이를 구현하는 기술인 JAXM을 통해 웹 서비스를 구성한다. JAXM은 역할에 따라 SAAJ와 JAXM 1.1_01로 구분된다.

- **SAAJ** : SAAJ는 SOAP 메시지를 생성시키고 위치시키는 역할을 한다. 즉, 정해져있는 SOAP 메시지의 포맷에 맞도록 LISI 저장소에 대한 LISI 데이타의 요청 및 응답을 구성하는데 필요한 API를 제공한다.
- **JAXM 1.1_01** : JAXM 1.1_01는 Message Provider를 이용한 SOAP 메시지를 비동기 방식으로 전송시키는 역할을 한다. 앞서 SAAJ가 SOAP 메시지를 구성하는 역할을 했다면 JAXM 1.1_01은 이 SOAP 메시지를 실제 전송하는 역할을 하는 것이다. 비동기 방식으로 전송하므로 하나의 Message Provider만 있어도 여러 개의 SOAP 메시지 전송이 가능하다. 한편 JAXM을 이용한 웹 서비스는 Message Provider를 통하여 로깅(Logging) 정보를 포함한 상태 감시 및 재전송을 실현하고, 또한 컨테이너에 실려 수행됨으로써 프로세스의 상시 동작을 지원 받도록 하여 신뢰성 있는 데이타 전달을 가능하도록 한다.

4. LISI 저장소의 프로토타입 구현

본 장에서는 논문에서 제시한 LISI 저장소 개발 프레임워크를 바탕으로 하여 실제 LISI 저장소의 프로토타입을 설계하고 구현하였다. LISI 저장소의 프로토타입은 성숙도 모델, 참조 모델, 능력 모델 및 구현옵션 테이블을 포함하고 있는 가장 중요한 LISI 데이타인 상호운용성 프로파일의 저장 및 검색 기능을 제공한다. 분석 및 설계의 프로세스로는 Unified Process를 확장하여 구체화한 방법인 분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법[27]을 이용하였다.

4.1 시스템 요구사항 추출 및 분석

그림 8은 LISI 저장소의 설계 방안 중 논리적 모델이 제공하는 구성 요소에 관한 정보를 바탕으로 생성한 유즈케이스 다이어그램이다. 저장소의 외부 구성 요소로서 Evaluation System(Inspector, LISI Reports)와 각 LISI 데이타 유형별 DB들(①)을 actor로 추출하고, 논리적 모델의 데이타 변환 모듈과 데이타베이스 접근 모듈(②)로부터 'Transform Info', 'Access DB' 등의 유즈케이스를 추출할 수 있다.

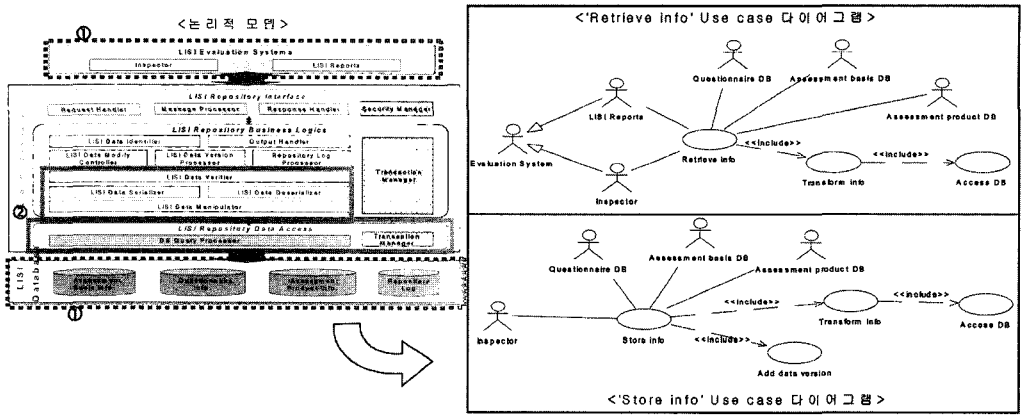

그림 8 'Retrieve info'와 'Store info'에 대한 유즈케이스 다이어그램

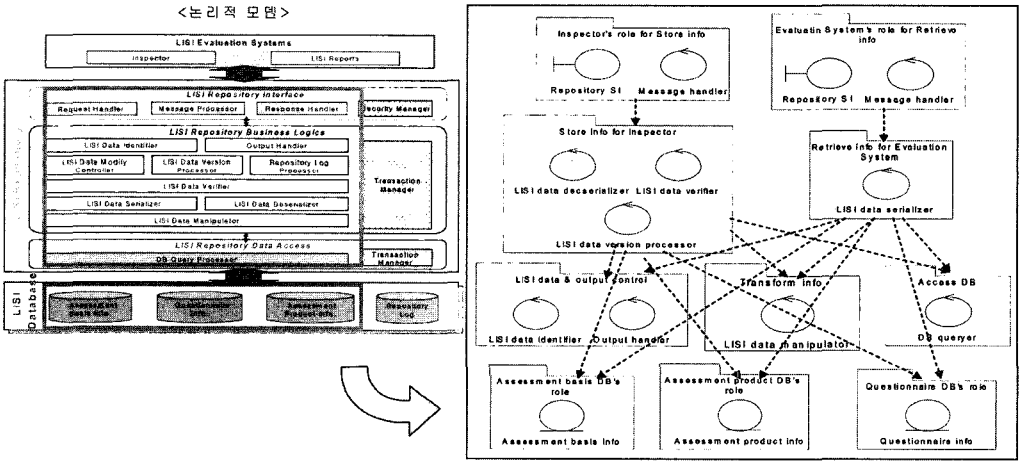

그림 9 분석 패키지 및 분석 클래스 추출

그림 9는 LISI 저장소의 기능적 요구 사항을 다루는 분석 클래스와 이를 의미 있는 단위로 묶은 분석 패키지를 추출한 형태이다. 설계 방안의 논리적 모델에서 나타내고 있는 인터페이스 층, 비즈니스 로직 층, 데이타 접근 층 및 LISI 데이타베이스의 내부 각 모듈로부터 저장소의 특성을 만족시키도록 3계층 구조에 따라 바운더리, 컨트롤, 엔티티 분석 클래스와 분석 패키지를 추출할 수 있다. 예를 들어, 논리적 모델의 인터페이스 층에 존재하는 Request Handler 모듈로부터 'Repository SI(System Interface)' boundary 분석 클래스와 'Message handler' control 분석 클래스를 추출할 수 있다.

그림 10은 앞서 추출한 분석 클래스들간의 상호 작용 관계를 LISI 저장소 설계 방안의 논리적 모델과 데이타 모델 정보를 바탕으로 생성한 'Retrieve info'에 대한 협력 다이어그램이다. 논리적 모델의 구성 모듈간 연관 관계와 데이타 모델의 입출력 데이타 및 프로세스를 통하여 각 분석 클래스의 메시지 호출을 정의할 수 있다. 하나의 예로, 그림 9에서 데이타 모델의 '검색요청정보 분석'으로부터 검색될 LISI 데이타의 유형 정보가 출력된다. 이러한 정보를 통해 협력 다이어그램의 'Message Handler' 분석 클래스는 'LISI data identifier' 분석 클래스에게 'identify data'라는 메시지를 보내고, 'LISI data identifier'는 'DB Queryer'에게 'Query' 메시지를 보내도록 분석할 수 있다.

4.2 시스템 설계

그림 11은 LISI 저장소 설계 방안의 물리적 모델과 데이타 모델의 정적 관계 정보를 이용하여 'Profile' 엔티티 빈에 대한 설계 클래스를 추출하고 관계를 정의한 클래스 다이어그램이다. 물리적 모델에서 추출한 'Profile' 엔티티 빈은 'ProrilePK', 'ProfileHome', 'Profile', 'ProfileEJB' 설계 클래스로 구성되며, CMP 필드로 들어갈 'systemId', 'revision', 'profileRefs'을 속성으로 가

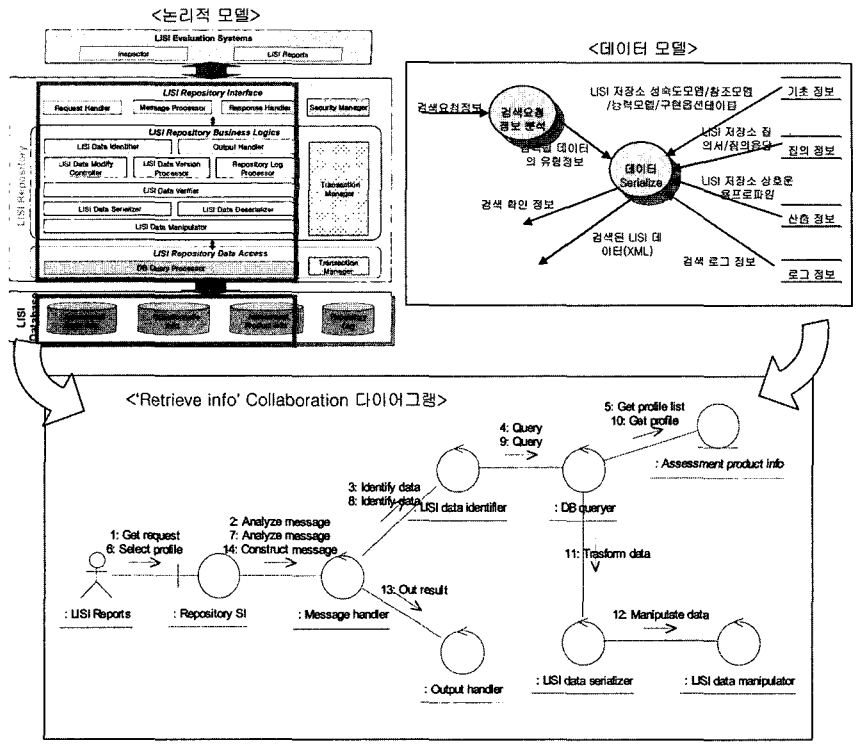

그림 10 'Retrieve info' 유즈케이스의 협력 다이어그램

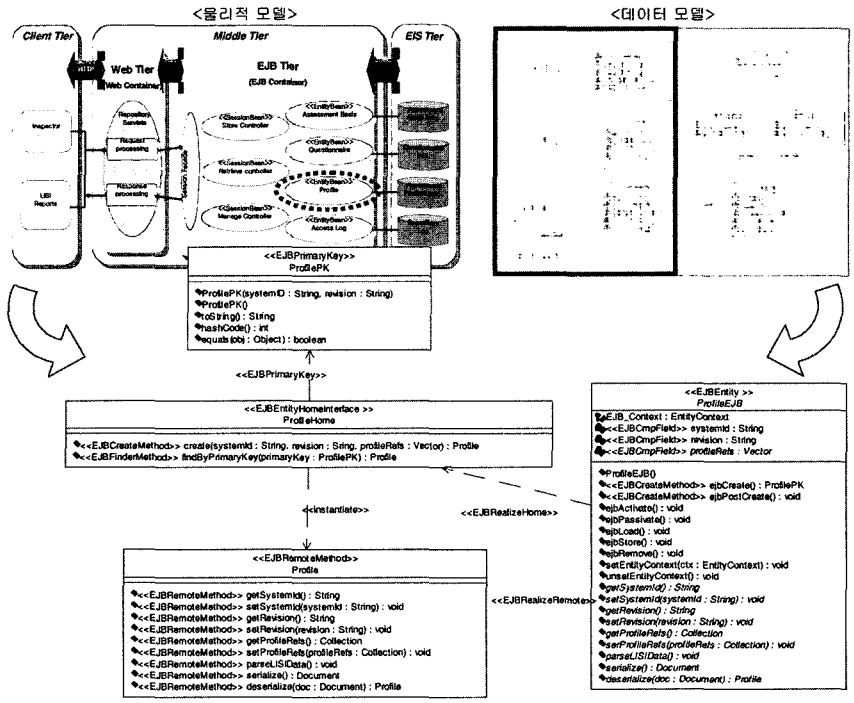

그림 11 'Profile'에 대한 클래스 다이어그램

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- capability97.xsd  -->
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <xsd:annotation>
      <xsd:documentation xml:lang="en">LISI capability Schema Example. Copyright 2002 SE Lab. All rights reserved.</xsd:documentation>
    </xsd:annotation>
    <xsd:element name="profile" type="profileType" />
  - <xsd:complexType name="profileType">
    - <xsd:sequence>
        <xsd:element name="procedure" type="interoperabilityType" maxOccurs="1" />
        <xsd:element name="application" type="interoperabilityType" maxOccurs="1" />
        <xsd:element name="infrastructure" type="interoperabilityType" maxOccurs="1" />
        <xsd:element name="data" type="interoperabilityType" maxOccurs="1" />
      </xsd:sequence>
      <xsd:attribute name="systemID" type="xsd:string" use="required" />
      <xsd:attribute name="revisionID" type="xsd:string" use="required" />
    </xsd:complexType>
  - <xsd:complexType name="interoperabilityType">
    - <xsd:sequence>
        <xsd:element name="implOpt" type="xsd:string" />
      </xsd:sequence>
      <xsd:attribute name="level" type="levelType" use="required" />
      <xsd:attribute name="subLevel" type="subLevelType" use="required" />
      <xsd:attribute name="capability" type="xsd:string" use="required" />
    </xsd:complexType>
  - <xsd:simpleType name="levelType">
    - <xsd:restriction base="xsd:string">
        <xsd:enumeration value="enterprise" />
        <xsd:enumeration value="domain" />
        <xsd:enumeration value="functional" />
        <xsd:enumeration value="connected" />
        <xsd:enumeration value="isolated" />
      </xsd:restriction>
    </xsd:simpleType>
  - <xsd:simpleType name="subLevelType">
    - <xsd:restriction base="xsd:string">
        <xsd:pattern value="[a-d]" />
      </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

그림 12 능력 모델의 XSD

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- edited by geonmo  -->
- <profile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="capability97.xsd" systemID="sys1" revisionID="rev1">
  - <procedure level="enterprise" subLevel="b" capability="Cross Government">
      <implOpt>implementation option 1</implOpt>
    </procedure>
  - <application level="functional" subLevel="c" capability="Web Browser">
      <implOpt>implementation option 2</implOpt>
    </application>
  - <infrastructure level="domain" subLevel="c" capability="WAN">
      <implOpt>implementation option 3</implOpt>
    </infrastructure>
  - <data level="enterprise" subLevel="a" capability="Enterprise Model">
      <implOpt>implementation option 4</implOpt>
    </data>
  </profile>
```

그림 13 예제 시스템의 XML 상호운용성 프로파일

진다. 이러한 속성들은 데이타 모델에서 정의한 LISI 데이타의 속성들을 바탕으로 추출할 수 있다. 그리고 필드에 대한 get/set 메쏘드와 LISI 데이타 변환 과정에서 필요한 'parseLISIData()', 그리고 XML 형태의 LISI 데이타를 ADM으로 변환하는 'serialize()', 역 과정의 'deserialize()' 메쏘드를 가진다. 물리적 모델에서의 'Profile' 엔티티 빈은 성숙도 모델, 참조 모델, 능력 모델 및 구현 옵션을 참조하기 위해 다른 엔티티 빈을 필요로 하며 이것은 'ProfileRef' 엔티티 빈이 지원한다.

4.3 시스템 구현

LISI 저장소의 구현 방안에서 제시한 구현 기술 통합 프레임워크를 바탕으로 지금까지 설계한 LISI 저장소의 프로토타입을 구현하였다. 구현된 LISI 저장소 프로토타입을 실행하여 임의의 상호운용성 평가체계의 LISI 데이타의 저장 및 검색 요청에 대한 서비스를 제공한다.

그림 12는 구현 방안에서의 LISI 저장소 구현 기술들 중 XSD를 이용하여 메타데이타의 성격을 가지는 능력 모델을 정의한 것이다. 그리고 이를 이용하여 임의의 평가 대상체계 예제 시스템의 상호운용성 프로파일을 그림 13에서와 같이 정의한다.

그림 14는 웹 컨테이너에 배치되어 저장소의 Request Processing, Response Processing의 기능을 겸하는 서블릿을 나타낸다. 이 서블릿은 JAXMServlet를 extend 함으로써 Inspector로부터 전송되어 오는 SOAP-Message를 onMessage() 메소드의 오버라이드를 통해 처리할 수 있다. 요청 서비스의 타입을 식별하여 저장일

```
// 서블릿 클래스
public class RequestPr cess r extends JAXMServlet implements OnewayListener {
    //...
    private Pr viderC nnecti nFact ry pcf;                          // Pr viderC nnecti n을 얻어오기 위한 팩토리
    private Pr viderC nnecti n pc;                  // 비동기적인 메시지 전송을 가능하게 하는 Pr viderC nnecti n
    private MessageFact ry mf = null;               // SOAPMessage 생성에 사용되는 메시지 팩토리
    // ...
    // SOAPMessage가 도착했을 때 서블릿 컨테이너에 의해 호출되는 meth d
    public v id  nMessage(SOAPMessage message) {
        try {
                SOAPPart sp = message.getSOAPPart();
                SOAPEnvel pe se = sp.getEnvel pe();
                SOAPHeader sh = se.getHeader();
                SOAPB dy sb = se.getB dy();
                String service_type = sb.getAttributeValue(se.createName("service_type"));
                if ("st re".equals(service_type)) {
                    sh.detachN de();
                    inputS urce is = SAXS urce.s urceT inputS urce(sp.getC ntent());      //SOAP 메시지의 b dy를 SAXS urce로 변환
                    LisiFacade lisiFacade = c nnectLisiFacade();
                    lisiFacade.st rePr file(is);                    // LisiFacade EJB bject의 st rePr file meth d 호출
                }
                else if ("retrieve".equals(service_type)) {
                    String sysID, revID, fr m, requestedURL;
                    fr m = "http://127.0.0.1:8000/lisi_rep sit ry/request_pr cess r";
                    sysID = sb.getAttributeValue(se.createName("sysID"));
                    revID = sb.getAttributeValue(se.createName("revID"));
                    requestedURL = sb.getAttributeValue(se.createName("requestedURL"));
                    LisiFacade lisiFacade = c nnectLisiFacade();
                    D cument d c = lisiFacade.retrievePr file(sysID, revID);   // LisiFacade EJB bject로부터 retrievePr file 호출
                    if (mf == null) {                                                  // SOAPMessage 팩토리를 생성
                            Pr viderMetaData metaData = pc.getMetaData();
                            String[] supp rtedPr files = metaData.getSupp rtedPr files();
                            String pr file = null;
                            //...
                            mf = pc.createMessageFact ry(pr file);
                    }
                    SOAPMessage msg = mf.createMessage();                              // 메세지 팩토리로부터 SOAPMessage를 생성
                    Element r   t = d c.getD cumentElement();
                    Element s ap_b dy = d .c.createElement("SOAP-ENV:B dy");
                    Element s ap_envel pe = d c.createElementNS("http://schemas.xmls ap. rg/s ap/envel pe/", "SOAP-ENV:Envel pe");
                    d c.rem veChild(r   t);
                    d c.appendChild(s ap_envel pe);
                    s ap_envel pe.appendChild(s ap_b dy);
                    s ap_b dy.appendChild(r   t);
                    SOAPPart part = msg.getSOAPPart();
                    part.setC ntent(new DOMS urce(d c));
                    msg.saveChanges();
                    SOAPRPMessageImpl s aprpMsg = new SOAPRPMessageImpl(msg);   // 메시지로부터 SOAPRP 구현 메시지 생성
                    s aprpMsg.setFr m(new Endp int(fr m));                             // 보낼 메시지에 대하여 송신자 URL 설정
                    s aprpMsg.setT  (new Endp int(requestedURL));                      // 보낼 메시지에 대하여 수신자 URL 설정
                    pc.send(s aprpMsg);                                                // Pr viderC nnecti n을 통하여 SOAPRP 메시지 전송
                }
                // ...
```

그림 14 LISI 저장소의 서블릿 클래스

경우 SOAP 메시지의 몸체(body)를 SAXSource로 변환하고 LisiFacade의 저장 메소드를 호출하며, 검색일 경우 검색 id를 식별하여 LisiFacade의 검색 메소드를 호출한다. 또한 OnewayListener를 구현함으로써 비동기 방식의 메시지 전달을 실현한다.

그림 15는 Profile 엔티티 빈 클래스이다. 모든 지속(persistence) 필드에 대해 get/set 메소드를 정의하고 EJB 생성 메소드를 정의한다. EJB 생성 메소드의 리턴 타입은 ProfilePK로 두어 EJB 컨테이너가 기본키 클래스로서 ProfilePK 클래스를 이용하도록 한다.

5. 결론 및 향후 연구

LISI 저장소는 체계간의 상호운용성 평가에 있어 효율적인 자료의 저장 및 관리를 지원하기 위한 필수적인 요소로서 평가체계와는 독립적이면서 표준화된 방법을 통해 그 기능을 제공할 수 있어야 한다. LISI 저장소의 비즈니스 로직들은 현재의 분산 환경에 맞도록 분산 컴포넌트로 개발되어져야 하며 표준화된 LISI 데이타의 표현이 요구된다. 이러한 LISI 저장소의 구축을 위해서는 LISI 저장소의 특성을 잘 반영할 수 있도록 기존의 소프트웨어 설계 방법을 보다 구체화할 필요가 있다. 뿐만 아니라 LISI 저장소의 구현에 필요한 다양한 구현 기술들을 인식하고 하나의 LISI 저장소의 구현에 통합 활용할 수 있도록 하는 방안이 필요하다. 본 논문에서는 상호운용성 평가 정보를 관리하는 LISI 저장소의 설계 및 구현에 있어서 LISI 저장소의 특성을 3가지 관점에서 고려하여 LISI 저장소에 구체화한 설계 방안과, 다양한 LISI 저장소 구현 기술을 하나의 LISI 저장소 구현을 위해 상호 유기적으로 통합 활용할 수 있는 구현 방안을 제시하였다. LISI 저장소의 설계 방안으로써는 LISI 저장소 설계 시 고려되어야 할 LISI 저장소의 특성 및 기능을 논리적 관점으로 분석한 논리적 모델과, 논리적 구성 요소들을 구현 기술에 특화된 형태로 정의한 물리적 모델, 그리고 LISI 데이타에 대해 LISI 저장

```
// Profile 엔티티 빈의 빈 클래스
public abstract class ProfileBean implements EntityBean {
    private EntityContext context;
    public abstract String getSysID();
    public abstract void setSysID(String sysID);
    public abstract String getRevID();
    public abstract void setRevID(String revID);
    public abstract String getLevelOfP();
    public abstract void setLevelOfP(String levelOfP);
    public abstract String getSubLevelOfP();
    public abstract void setSubLevelOfP(String subLevelOfP);
    //...
    // EJB 생성 method.
    public ProfilePK ejbCreate(String sysID, String revID, String levelOfP, String subLevelOfP, String capabilityOfP,
                String implOptOfP, String levelOfA, String subLevelOfA, String capabilityOfA, String implOptOfA, String levelOfI,
                String subLevelOfI, String capabilityOfI, String implOptOfI, String levelOfD, String subLevelOfD,
                String capabilityOfD, String implOptOfD) throws CreateException {
                setSysID(sysID);
                setRevID(revID);
                setLevelOfP(levelOfP);
                setSubLevelOfP(subLevelOfP);
                setCapabilityOfP(capabilityOfP);
                // ...
                return null;
    }

    public void ejbPostCreate(String sysID, String revID, String levelOfP, String subLevelOfP, String capabilityOfP,
                String implOptOfP, String levelOfA, String subLevelOfA, String capabilityOfA, String implOptOfA, String levelOfI,
                String subLevelOfI, String capabilityOfI, String implOptOfI, String levelOfD, String subLevelOfD,
                String capabilityOfD, String implOptOfD) throws CreateException {}
    public void ejbRemove() {}
    public void setEntityContext(EntityContext ctx) {
                this.context = ctx;
    }
    public void unsetEntityContext() {
                this.context = null;
    }

    // 나머지 컨테이너에 의해 호출되는 method는 구현하지 않는다.
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate(){}
    public void ejbPassivate(){}

    // 엔티티 빈이 가지고 있는 persistent data를 가져올 때 사용되는 business method
    public ProfileDetails getProfileDetails(){
                ProfileDetails pDetails = new ProfileDetails(getSysID(), getRevID(), getLevelOfP(), getSubLevelOfP(), getCapabilityOfP(),
                getImplOptOfP(), getLevelOfA(), getSubLevelOfA(), getCapabilityOfA(), getImplOptOfA(), getLevelOfI(), getSubLevelOfI(),
                getCapabilityOfI(), getImplOptOfI(), getLevelOfD(), getSubLevelOfD(), getCapabilityOfD(), getImplOptOfD());
                return pDetails;
    }
}
```

그림 15 Profile 엔티티 빈의 빈 클래스

소에서의 동적 흐름과 정적 관계를 정의한 데이타 모델을 제시하였다. LISI 저장소의 구현 방안으로서는 LISI 데이타 표현을 위한 기술과 분산 환경에서 컴포넌트 기반의 개발을 가능하게 하는 분산 지원 기술, 웹 환경에서 평가체계 어플리케이션과의 연결 수단을 제공하는 웹 서비스 기술의 역할과 각 기술간의 상호 관계를 정의함으로써 LISI 저장소의 통합 운용을 지원할 수 있는 구현 프레임워크를 제시하였다. 마지막으로, 제시한 LISI 저장소 개발 프레임워크를 바탕으로 실제 프로토타입을 구현해 봄으로써, 다양한 관점에서의 실질적인 설계와 구현 기술의 통합 활용에 대한 가이드의 유용성을 확인하였다. 본 논문에서 제시한 LISI 저장소 개발 프레임워크는 실제 설계자 및 개발자들에게 LISI 저장소의 구축 시 고려되어야 할 요소와 적용 방법들에 대한 구체적인 가이드라인을 제시해줌으로써 보다 쉽고 효과적인 LISI 저장소의 구축을 돕는다.

향후에는 본 논문에서 제시한 LISI 저장소의 설계 및 구현 방안을 적용하여 LISI 저장소 프로토타입을 확장, 개발하는 연구가 필요하다. 그리고 LISI 저장소 구축 방안을 보완하여 보다 다양한 LISI 데이타의 관리 기능을 제공하고, LISI 저장소의 성능 측면에서의 고려사항을 추가한 구축 방안 연구가 필요하다.

참 고 문 헌

[1] 로코즌(주) 부설 객체 연구소, 공통운용환경(COE) 구축지침 및 조문화, 2001.

[2] C4ISR Architecture Working Group, *Levels of Information Systems Interoperability (LISI)*, 1998.

[3] W3C, "Extensible Markup Language(XML) 1.0 (Second Edition)," October 2000.

[4] Sun Microsystems, *J2EE 1.4 Specification*, http://j2ava.sun.com/j2ee.

[5] Sun Microsystems, *Enterprise JavaBeans 2.0 Specification*, http://java.sun.com/product/ejb.

[6] Jag Sodhi and Prince Sodhi, Software Reuse: domain analysis and design process, McGraw- Hill, 1999.

[7] Guo, J. and Luqi, "A Survey of Software Reuse Repositories," Seventh IEEE International Confer-

ence and Workshop on Engineering of Computer Based Systems(ECBS2000), pp.92-100, 2000.
[8] Elisabetta Morandin, "SALMS v5.1: A System for Classifying, Describing, and Querying about Reusable Software Assets," The Proceedings of 5th International Conference on Software Reuse(ICSR '98).
[9] ComponentSource, http://www.componentsource.com
[10] 컴포넌트 뱅크, http://www.component-bank.com
[11] OMG, *OMG CORBA Component Model specification version 3.0*, June 2002.
[12] Box, D., *Essential COM*, Addison-Wesley, January 1998.
[13] SEI in CMU, "Component-Based Software Development/COTS Integration," http://www.sei.cmu.edu/str/descriptions/cbsd_body.html.
[14] OMG, *OMG Unified Modeling Language Specification, version 1.4*, September 2001.
[15] D'Souza, D. and Wills, *A.C., Objects, Frameworks, and Components with UML*, Addison-Wesley, 1998.
[16] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, January 1999.
[17] Sterling, *The CBD96 Standard Version 2.1*, Sterling, July 1998.
[18] Harmon, P., "Visual Modeling Tools, Case Vendors, and Component Methods," Component Development Strategies, pp. 1-16, June 1999.
[19] MSDN Online Library, http://www.microsoft.com/korea/msdn/library.
[20] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," IEEE Internet Computing, 2002.
[21] Vaughan-Nichols, S.J., "Web Services: Beyond the Hype," IEEE Computer, Vol. 35, Issue. 2, February 2002.
[22] Deitel, H.M. and Deitel, P.J., *XML How To Program*, Prentice Hall, 2001.
[23] Sun Microsystems, "The Java Enterprise Edition Developers Guide," 2000.
[24] Roman, E., *Mastering Enterprise JavaBeans 2nd Edition*, Wiley, 2002.
[25] Fisher, M. and Bodoff, S., *The Java Web Services Tutorial*, Sun Microsystems, 2002.
[26] 안성아, 최희석, 염근혁, "J2EE 어플리케이션 모델 기반의 컴포넌트 저장소 구현", 한국멀티미디어학회, 제5권 제1호, pp. 77-93, 2002.
[27] 최유희, 염근혁, "분산 컴포넌트 기반의 소프트웨어 분석 및 설계 방법", 한국정보과학회 논문지(소프트웨어 및 응용), 제28권 제12호, pp. 896-909, 2001.

조 정 희
2002년 2월 부산대학교 컴퓨터공학과(학사). 2004년 2월 부산대학교 컴퓨터공학과(석사). 2004년 3월~현재 국방과학연구소. 관심분야는 임베디드 시스템, 컴포넌트 기반 소프트웨어 개발, 분산 소프트웨어 시스템, 소프트웨어 아키텍처 등임

정 병 훈
2001년 2월 부산대학교 컴퓨터공학과(학사). 2003년 2월 부산대학교 컴퓨터공학과(석사). 2003년 3월~현재 LG정보통신
관심분야는 소프트웨어 아키텍처, 소프트웨어 개발 프로세스, 분산 환경 기반의 소프트웨어 설계 및 구현 등임

염 근 혁
정보과학회논문지 : 소프트웨어 및 응용
제 31 권 제 8 호 참조