# 온톨로지 기반 데이터 가변성 처리 기법
## (An Ontology-based Data Variability Processing Method)

임 윤 선 † 김 명 ††
(Yoonsun Lim) (Myung Kim)

**요 약** 다계층 구조를 갖는 현대의 기업용 분산 애플리케이션에서 비즈니스 엔티티는 로직을 구현한 각 계층의 서비스 컴포넌트들을 관통하는 일종의 횡단관심사이다. 비즈니스 엔티티가 변화하면 이와 관련된 서비스 컴포넌트들은 비록 구현된 기능이 바뀔 필요가 없을지라도 새로운 타입의 비즈니스 엔티티를 다룰 수 있도록 수정되어야 한다. 본 연구팀은 이전 연구에서 서비스 컴포넌트로부터 외부화된 데이터인 비즈니스 엔티티에 대한 가변성을 효율적으로 처리하기 위한 DTT 컴포넌트 모델(Data Type-Tolerant Component Model)을 제안하였다. DTT 컴포넌트 모델은 서비스 컴포넌트들과 비즈니스 엔티티들 간의 직접적인 결합을 없앰으로써 서비스 컴포넌트들이 수정되지 않고도 새로운 비즈니스 엔티티들을 처리할 수 있게 된 반면, 이들을 중재하는 데이터 타입 컨버터를 개발해야 하는 부담이 발생한다. 이에 본 논문에서는 서비스 컴포넌트의 SCDT(Self-Contained Data Type)와 비즈니스 엔티티의 각 속성에 대한 메타데이터로 온톨로지를 사용하는 방법과, 이를 이용하여 데이터 타입 컨버터 코드를 생성하는 방법을 제안한다. 본 논문에서 제안하는 온톨로지 기반 DTT 컴포넌트 모델은 컴퓨터가 에러 없이 데이터 타입 컨버터를 자동으로 생성할 수 있게 함으로써, 서비스 컴포넌트들의 재사용성과 데이터 가변성 처리 효율을 크게 향상시킨다.

**키워드** : 온톨로지, 데이터 가변성, 데이터 타입 컨버터, DTT 컴포넌트 모델, 프로덕트 라인

***Abstract*** In modern distributed enterprise applications that have multilayered architecture, business entities are a kind of crosscutting concerns running through service components that implements business logic in each layer. When business entities are modified, service components related to them should also be modified so that they can deal with those business entities with new types, even though their functionality remains the same. Our previous paper proposed what we call the DTT (Data Type-Tolerant) component model to efficiently process the variability of business entities, which are data externalized from service components. While the DTT component model, by removing direct coupling between service components and business entities, exempts the need to rewrite service components when business entities are modified, it incurs the burden of implementing data type converters that mediate between them. To solve this problem, this paper proposes a method to use ontology as the metadata of both SCDTs (Self-Contained Data Types) in service components and business entities, and a method to generate data type converter code using the ontology. This ontology-based DTT component model greatly enhances the reusability of service components and the efficiency in processing data variability by allowing the computer to automatically generate data type converters without error.

**Key words** : Ontology, Data Variability, Data Type Converter, Component Model, Product Line

· 이 논문은 2007년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원을 받아 연구되었음(KRF-2007-313-D00654)

† 학생회원 : 이화여자대학교 컴퓨터학과
lys96@ewhain.net
†† 종신회원 : 이화여자대학교 컴퓨터학과 교수
mkim@ewha.ac.kr

논문접수 : 2009년 10월 8일
심사완료 : 2010년 2월 9일



정보과학회논문지: 소프트웨어 및 응용 제37권 제4호(2010.4)

## 1. 서 론

소프트웨어 프러덕트 라인 공학이나(PLE: Product Line Engineering) 애플리케이션 프레임워크에서 가변성(variability) 처리 효율은 애플리케이션 개발 생산성과 직결되는 중요한 문제이다. 또 애플리케이션이 프러덕트 라인이나 애플리케이션 프레임워크를 기반으로 만들어지지 않았다 하더라도 시간이 흐르면서 발생하는 변화를 기민하게 수용하기 위해 가변성을 효율적으로 처리할 필요가 있다. 현대의 기업 애플리케이션 개발에서 주로 채택되는 다계층 아키텍처(multi-layered architecture)와 애플리케이션 서버(application server) 같은 미들웨어는 성능이나 기능요구 사항의 변화와 같은 가변성을 쉽게 다룰 수 있도록 해준다. 그러나 다계층 아키텍처에서 특징적으로 나타나는 비즈니스 엔티티(business entity), 즉 서비스 컴포넌트로부터 외부화(externalized)된 데이터에 대한 가변성을 효율적으로 처리하기 어렵다. 비즈니스 엔티티는 로직을 구현한 각 계층의 서비스 컴포넌트들을 관통하는 일종의 횡단관심사(cross-cutting concerns)이므로 비즈니스 엔티티가 변화하면 이와 관련된 서비스 컴포넌트들은 비록 구현된 기능이 변경 될 필요가 없을지라도 새로운 타입의 비즈니스 엔티티를 다룰 수 있도록 모두 수정되어야 하기 때문이다.

본 연구팀은 [1]에서 외부화된 데이터인 비즈니스 엔티티에 대한 가변성을 효율적으로 처리하기 위한 DTT 컴포넌트 모델(Data Type-Tolerant Component Model)을 제안하였다. DTT 컴포넌트 모델은 서비스 컴포넌트들과 비즈니스 엔티티들 간의 직접적인 결합을 없애기 위해, 서비스 컴포넌트의 코드는 비즈니스 엔티티들을 직접 다루지 않고, 이를 대신하는 자체 데이터 타입(SCDT: Self-Contained Data Type)만을 액세스(access)한다. 이는 서비스 컴포넌트들을 수정하지 않고도 새로운 비즈니스 엔티티들을 처리할 수 있게 한 반면, SCDT와 비즈니스 엔티티를 중재하는 데이터 타입 컨버터를 생성해야 하는 새로운 부담을 야기시킨다. DTT 컴포넌트 모델에서 데이터 타입 컨버터 생성 작업의 효율을 높이기 위해 SCDT와 비즈니스 엔티티의 각 속성들에 의미를 기술하는 자연어 기반의 메타데이터를 부가할 수 있지만, 이 경우 자연어가 가지는 모호성(ambiguity)때문에 데이터 타입 컨버터 코드 작성 시 에러를 유발할 수 있고 수작업에 의존하므로 변화에 대한 대응이 지연될 수 있다.

본 논문에서는 DTT 컴포넌트 모델의 SCDT와 비즈니스 엔티티의 각 속성에 부가하는 DTT 속성 메타데이터로 온톨로지를 사용하고, 컴포넌트 조립 시 이 메타데이터를 이용해 데이터 타입 컨버터를 자동 생성하는 온톨로지 기반 DTT 컴포넌트 모델을 제안한다. 제안하는 DTT 속성 메타데이터는 SCDT와 비즈니스 엔티티의 각 속성들의 의미와 값의 형식, 변환 기준을 자연어 대신 도메인 온톨로지와 확장된 애플리케이션 온톨로지를 사용하여 기술하므로, 컴퓨터가 이들을 해석하여 에러 없이 자동으로 데이터 타입 컨버터 코드를 생성할 수 있다. 서비스 컴포넌트들과 비즈니스 엔티티 결합을 없애고, 데이터 타입 컨버터 코드 생성을 자동화한 온톨로지 기반 DTT컴포넌트 모델은 소프트웨어 프러덕트 라인 개발이나 기업 애플리케이션의 유지 보수에서 데이터 가변성을 매우 효율적으로 처리할 수 있게 한다.

본 논문은 다음과 같이 구성된다. 2장에서 관련 연구들을 살펴보고, 3장에서는 본 논문의 기반 기술인 DTT 컴포넌트 모델에 대해 설명한다. 4장에서 온톨로지 기반 DTT 컴포넌트 모델을 정의하고, 데이터 타입 컨버터 자동 생성 방법을 제안한다. 5장에서는 학점 체계가 다른 대학과 새롭게 학점 교류 협약을 맺더라도 평점 계산 소프트웨어를 수정하지 않고 데이터 가변성을 처리하는 구체적인 예를 통해 본 논문에서 제시한 기법을 자세히 설명하고, 6장에서 결론을 맺는다.

## 2. 관련 연구

본 장에서는 기업 애플리케이션 모델의 특징적으로 나타나는 데이터 가변성과 기존의 가변성 처리 연구들을 살펴본다. 기업 애플리케이션에서 비즈니스 엔티티들은 각 계층의 서비스 컴포넌트들을 관통하는 일종의 횡단 관심사이기 때문에, 이와 같은 데이터 가변성을 효과적으로 처리하려면 비즈니스 엔티티 자체의 변화와 더불어 이에 따른 서비스 객체들의 이차적인 변화까지도 효율적으로 수용하는 기법이 필요하다. 기존의 가변성 처리 연구들에서는 직접적으로 변화하는 객체만을 대상으로 하는 일차적인 가변성 문제만 다루거나, 애플리케이션 구성 객체의 메타데이터를 런타임에 해석하여 처리함으로써 가변성을 유연하게 처리하지만 개발 비용이 과도하게 요구되고 성능상 문제가 있거나, 동적으로 가변성을 수용하면서 주로 의미론적 매치 문제만 집중하고 구체적인 변환 기법이 생략된 것들이 있다.

### 2.1 기업 애플리케이션 아키텍쳐 스타일

하드웨어와 네트워크 성능이 비약적으로 발전하고 위치 투명성을 지원하는 컴포넌트 기술이 등장하면서 다계층 구조가 현대의 기업 애플리케이션의 일반적인 아키텍처 스타일로 자리 잡았다. 기업 애플리케이션 구현 기술인 Java Enterprise Edition이나 닷넷의 참조 모델에서 프리젠테이션 로직이나 비즈니스 로직을 처리하는 서비스 컴포넌트들과 비즈니스 데이터인 비즈니스 엔티티를 분리하고 있다[2,3]. 애플리케이션이 동작할 때 각

계층에 자리잡은 서비스 컴포넌트들의 메소드들은 입/출력 매개 변수로 비즈니스 엔티티 객체들을 주고 받아 그들의 속성들을 액세스하기 때문에, 비즈니스 엔티티가 변하면 이들이 관통하는 많은 서비스 컴포넌트들도 수정해야 한다. 객체지향 옹호론자들은 속성들만 가지는 비즈니스 엔티티 대신 행동 특징(behavioral feature)도 함께 구현한 도메인 객체(domain object)를 사용해야 한다고 주장하지만[4,5], 도메인 객체 역시 다른 계층의 서비스 객체들에게 입출력 매개 변수로 전달되고 그 속성이 액세스되어야 하는 비즈니스 데이터이다. 따라서 기업 애플리케이션의 유지 보수 효율을 높이기 위해서, 또는 효율적인 프러덕트 라인이나 애플리케이션 프레임워크를 만들기 위해서는 서비스 컴포넌트와 비즈니스 데이터 간 결합을 완화시켜서 데이터 가변성을 효율적으로 처리하는 연구가 필요하다.

### 2.2 아키텍처 기반 가변성 처리 연구

가변성을 모델링하는 방법으로 패턴을 사용하는 연구[6], UML 확장 기법을 이용하는 연구[7]가 있다. 또 정보 은닉, 상속, 가변점을 사용하는 연구[8]도 있다. 이들은 모두 객체지향의 특성을 기반으로 한 연구들로서 아키텍처 수준에서 컴포넌트들을 선택 혹은 대체하는 형태의 기능적 가변성이나 로컬화된 데이터의 가변성만 다루고 있다. 그러나 이들은 아키텍처 수준에서 가변점(variation point)에 대응하는 직접적인 가변치들을 모델링의 요소로 나타낼 뿐 애플리케이션 아키텍처에 새로 도입되는 가변치들에 의해서 영향 받는 연관된 요소에서의 이차적인 가변성을 표현하지 못한다. 현대의 기업 애플리케이션에서 재사용될 수 있는 프로덕트 라인을 구축하기 위해서는 기업마다 변화하는 비즈니스 엔티티에 대한 가변성 처리와 더불어 외부화된 데이터의 변화가 서비스 컴포넌트들에 미치는 이차적인 가변성을 효과적으로 처리할 수 있는 연구가 필요하다.

### 2.3 메타데이터 기반 가변성 처리 연구

기업 애플리케이션의 비즈니스 변화 요구를 재코딩 없이 동적으로 수용하기 위해 메타데이터를 이용하는 연구들[9,10]이 있다. 대표적인 메타데이터 지향 아키텍처인 AOM(Adaptive Object Model)에서는 TypeObject, Properties, Type Square, Entity-Relationship, Rule-Object등의 메타데이터를 통해 애플리케이션의 객체들과 그들의 속성들 그리고 행위들을 표현하고 런타임에 이 메타데이터를 해석하여 객체들을 생성하고 동작시킨다. AOM은 비즈니스 요구 사항이 바뀔 때 클래스 구현 코드를 수정하는 대신 메타데이터들만 수정하면 인터프리터(interpreter)가 바뀐 객체를 생성해주기 때문에 매우 효율적으로 가변성을 처리할 수 있다. 그러나 효율적으로 가변성을 처리하는 시스템을 만들기 위해서는 복잡한 메타데이터들을 잘 설계하고 관리해야 하며 보통의 애플리케이션 개발에서는 필요 없는 객체 생성 인터프리터 등을 구현해야 한다. AOM기반 개발은 과도한 초기 구축 비용이 요구되므로 프로덕트 라인 개발 등에서 고려할 수 있지만 일반적인 애플리케이션 개발에는 적합하지 않다. 따라서 개발 효율성과 유지 보수 효율성이 모두 높은 가변성 처리 기술이 필요하다.

### 2.4 온톨로지 기반 동적 서비스 조립 연구

온톨로지를 활용하여 서비스 들을 동적으로 조립하고 호출하는 방법에 관한 연구들[11-13]이 있다. 동적으로 관련된 서비스들을 찾아 이들을 조립하고 호출하려면 서비스들의 입/출력 매개변수간 정합을 위하여 중재(mediation) 코드 자동 생성이 필수적이다. 파라미터간 정합을 위한 중재 코드 자동 생성에 온톨로지를 활용하는 것은 본 논문의 데이터 타입 컨버터 코드 자동 생성 방법과 유사한 목표이다. 웹 서비스에 대한 발견과 자동 조립 및 호출 지원을 표방하는 OWL기반의 서비스 온톨로지 언어인 OWL-S[11]는 Service Profile, Service Model, Service Grounding 온톨로지로 구성되어 있다. 그러나 이들은 조립과 호출에 필요한 메타 정보들에 관한 규정일 뿐 구체적인 조립과 호출 알고리즘이나 방법을 제안하지 않는다. 사용자의 질의를 분석하여 동적으로 관련 서비스 컴포넌트들을 찾아 조립하고 호출하여 답을 구하는 것을 목표로 하는 동적 서비스 조립(dynamic service composition)에서도 동적 서비스 조립과 호출이라는 목표를 달성하려면 서비스 간 파라미터 정합 코드 생성이 필수적이다. 그러나 관련 연구들[12,13]에서는 의미를 기준으로 매칭되는 서비스와 매개 변수들을 찾는데 온톨로지를 활용하는 방법을 제시하는데 중점을 두고, 매개 변수들의 정합 코드 생성에 온톨로지를 활용하는 방법을 제시하지 않고 있다.

## 3. DTT(Data Type-Tolerant) 컴포넌트 모델

본 연구팀은 이전 연구에서 서비스 컴포넌트로부터 외부화된 데이터인 비즈니스 엔티티에 대한 가변성을 효율적으로 처리하기 위한 DTT 컴포넌트 모델을 제안하였다[1]. 본 장에서는 DTT 컴포넌트 모델을 간략히 살펴본다. DTT 컴포넌트 모델은 서비스 컴포넌트와 비즈니스 엔티티간 결합을 완화시키기 위해, 서비스 컴포넌트의 코드에서 비즈니스 엔티티들을 직접 다루지 않고, 이를 대신하는 자체 데이터 타입(SCDT: Self-Contained Data Type)만을 액세스한다. 서비스 컴포넌트는 실제로 다루어야 할 비즈니스 엔티티와 SCDT를 연결해주는 가변점 인터페이스를 같이 정의하여 내포한다. 이 인터페이스는 서비스 컴포넌트들과 비즈니스 엔티티들을 조립할 때 구현되는데, 구현 코드는 SCDT와 비즈

니스 엔티티 속성 간 변환이 필요할 때 이를 처리하는 코드들을 포함할 수 있으므로 데이터 타입 컨버터라 부른다. DTT 컴포넌트 모델에서 서비스 컴포넌트들과 비즈니스 엔티티 간 결합을 완화시키는 역할을 담당하는 데이터 타입 컨버터들은 애플리케이션 실행 시 DTT 프레임워크에 의해 객체화되어 의존성 주입 방법으로 SCDT의 속성에 할당된다.

그림 1은 DTT 컴포넌트 모델 구조이다. DTT 컴포넌트 모델은 애플리케이션의 비즈니스 로직을 구현한 서비스 컴포넌트, 비즈니스 데이터 역할을 담당하는 비즈니스 엔티티, 비즈니스 엔티티를 서비스 컴포넌트의 SCDT로 변환하는 데이터 타입 컨버터 컴포넌트로 구성된다.

**루트 데이터 타입 / 비즈니스 엔티티**

루트데이터 타입은 DTT 컴포넌트 모델에서 모든 비즈니스 엔티티 타입이 상속해야 하는 기본 데이터 타입이다. 서비스 컴포넌트는 구성 메소드의 매개 변수나 반환 값을 루트 데이터 타입으로 선언함으로써 모든 비즈니스 엔티티를 수용할 수 있게 된다. 루트 데이터 타입은 명칭만 가지는 형식으로 정의되고, 모든 개발자들에게 기본 클래스 라이브러리로 제공된다.

**서비스 컴포넌트 구조**

서비스 컴포넌트는 컴포넌트 기능을 구현한 서비스 객체 외에, 실제로 처리해야 할 애플리케이션의 비즈니스 엔티티 대신하는 SCDT와, 비즈니스 엔티티 타입을 SCDT로 변환하기 위해 구현할 데이터 타입 컨버터 객체의 모태인 데이터 타입 컨버터 가변점 인터페이스를 추가로 내장하는 구조를 갖는다.

**데이터 타입 컨버터 (Data Type Converter) 컴포넌트 구조**

데이터 타입 컨버터 컴포넌트는 서비스 컴포넌트들과 비즈니스 엔티티 컴포넌트들을 조립할 때 생성되는 가변점 인터페이스의 구현체인데, 애플리케이션 실행 시 비즈니스 엔티티를 서비스 컴포넌트의 SCDT로 타입을 변환해준다. 데이터 타입 컨버터 객체는 SCDT의 각 속성에 대하여 Get, Set 액세서(accessor)를 구현하며, 서비스 객체가 SCDT의 속성에 액세스할 때마다 해당 메소드가 호출된다. 각 메소드에서는 SCDT 속성에 대응되는 비즈니스 엔티티의 속성값을 읽어 SCDT 속성값으로 변환하는 로직을 실행한다. 데이터 타입 컨버터는 서비스 컴포넌트와 비즈니스 엔티티들을 조립할 때 생성해야 하는데, 이 작업의 효율을 높이기 위해 SCDT와 비즈니스 엔티티를 구성하고 있는 속성들에게 그 의미를 기술하는 DTT 속성 메타데이터를 부가할 수 있다.

## 4. 온톨로지 기반 DTT 컴포넌트 모델

3장에서 설명한 DTT 구성 요소 중 데이터 타입 컨버터는 서비스 컴포넌트가 비즈니스 엔티티대신 SCDT를 액세스할 때 비즈니스 엔티티 객체의 속성을 그에 대응하는 SCDT 속성으로 변환해주는 타입 변환 메소드들로 구성된다. 컴포넌트 조립시 데이터 타입 컨버터 생성 작업의 효율을 높이기 위해 SCDT와 비즈니스 엔티티의 각 속성에 대하여 그 의미를 기술하는 자연어 기반의 메타데이터를 부가할 수 있다. 그러나 자연어가 가지는 모호성은 데이터 타입 컨버터 코드 작성 시 에러를 유발할 수 있고, 수작업에 의존하기 때문에 환경 변화에 대해 신속하게 대응할 수 없다. 따라서 본 장에서는 데이터 타입 컨버터를 에러 없이 자동으로 생성하기 위해 도메인 온톨로지와 확장된 애플리케이션 온톨로지를 DTT 속성 메타데이터로 사용하는 온톨로지 기반 DTT 컴포넌트 모델을 제안한다.

### 4.1 데이터 가변성 처리를 위한 온톨로지 모델

본 논문에서는 애플리케이션의 정보 모델을 온톨로지 언어인 OWL(Web Ontology Language)로 기술할 것을 제안한다. 기업에서 사용하는 소프트웨어는 주로 정보를 처리하는 시스템이기 때문에, 기업 애플리케이션

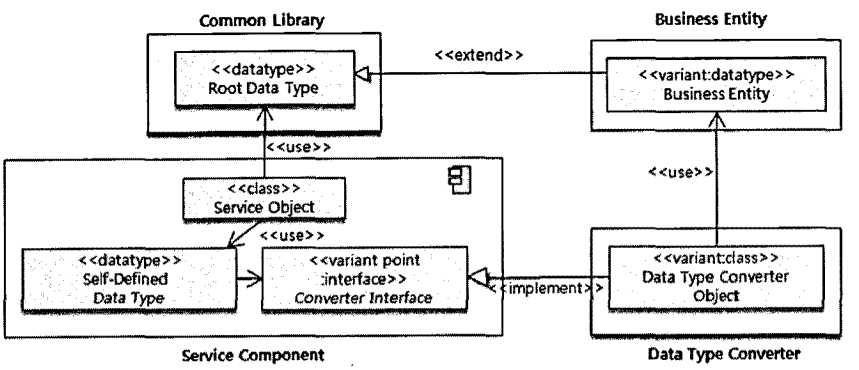


그림 1 DTT 컴포넌트 모델

개발 프로세스의 초기 과정인 분석단계에서 정보들을 정확하게 식별하고 기술하는 것이 매우 중요하다. 정보 모델은 보통 클래스 다이어그램이나 ER 다이어그램으로 표현하고 자연어 기반의 설명을 덧붙여서 작성하는데, 이런 기술 방법으로는 정보를 정확하고 정밀하게 표현하기 어렵기 때문에 현업 사용자, 아키텍트, 개발자, 테스터 등 각 이해 관련자들간 오해가 발생할 소지가 많다. 특히 개발 프로세스의 초기 단계에서의 오류나 오해를 바로잡으려면 많은 비용이 필요하다. 이에 반해 OWL과 같은 온톨로지 언어를 사용하여 정보 모델을 기술하면 클래스 다이어그램과 같이 개념, 속성, 관계 등을 표현할 수 있을 뿐 아니라 개념과 속성의 표현에 제약 조건이나 필요/충분 조건의 형식적인 공리(formal axiom)들을 함께 기술함으로써 정보를 정확하고 정밀하게 표현할 수 있으므로 이해 관련자간 오해를 제거할 수 있고 기계 해석이 가능해진다.

그림 2는 정보 표현에 온톨로지 모델을 사용하는 온톨로지 기반 데이터 가변성 처리 기법을 적용한 개발 프로세스를 보여준다. 분석 단계에서 도출된 도메인 온톨로지 모델은 설계 단계에서 애플리케이션 고유의 정보들을 반영한 애플케이션 온톨로지 모델로 확장되고, 데이터베이스 스키마와 비즈니스 엔티티 클래스 설계에 사용된다. 컴포넌트 구현 단계에서 온톨로지는 각 인터페이스 오퍼레이션의 매개 변수들과 내장 데이터 타입, 비즈니스 엔티티들의 속성에 대한 의미 기반 메타데이터로 사용된다. 컴포넌트 조립 단계에서 조립도구는 비즈니스 엔티티 컴포넌트들과 서비스 컴포넌트들을 조립할 때 온톨로지를 사용하여 이들을 중재할 때 필요한 데이터 타입 컨버터 코드를 자동 생성한다.

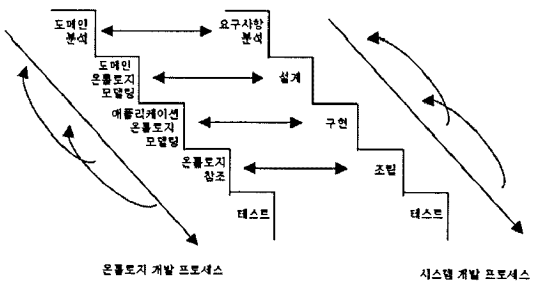


그림 2 온톨로지 기반 시스템 개발 프로세스

### 4.2 온톨로지 기반 DTT 컴포넌트 메타모델

그림 3은 온톨로지 기반 DTT 컴포넌트 메타 모델이다. 온톨로지 기반 컴포넌트 모델은 3장에서 설명한 DTT 컴포넌트 모델의 핵심 구성 요소인 SCDT와 타입 변환 인터페이스, 비즈니스 엔티티와 SCDT 속성간 구문, 의미의 불일치를 중재하는 데이터 타입 변환 커넥

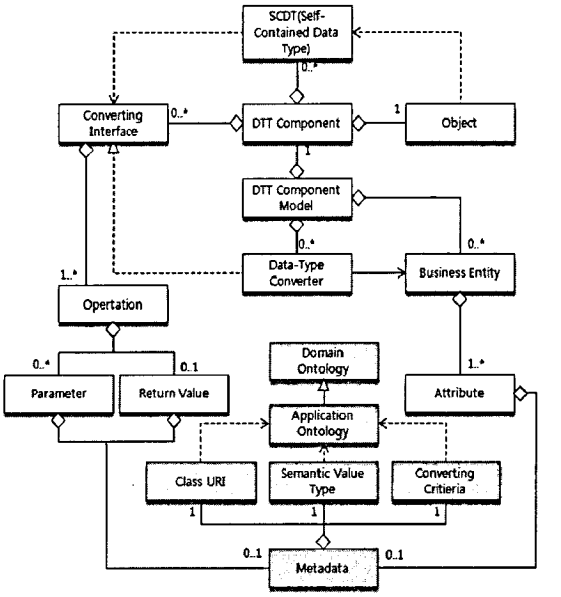


그림 3 온톨로지 기반 DTT 컴포넌트 모델의 메타 모델

터와 더불어, 타입 변환 인터페이스와 비즈니스 엔티티에 대해 새롭게 부가된 온톨로지 기반 메타데이터로 구성된다. DTT 컴포넌트 모델에서 데이터 가변성을 처리하는 데이터 변환 커넥터를 에러 없이 개발하기 위해 자연어 기반의 설명이나, 레퍼런스 매뉴얼, 샘플 코드 등을 작성하던 것을, 온톨로지 기반 DTT 컴포넌트 모델에서는 타입 변환 인터페이스의 각 매개변수들과 리턴 값, 비즈니스 엔티티의 각 속성(변수)들에 온톨로지 기반 메타데이터를 부가하는 것으로 대신한다. 온톨로지 기반 DTT 컴포넌트 모델의 메타데이터는 각 변수들의 의미, 변수가 취하는 값, 타입 변환 기준을 4.1절에서 설명한 도메인 정보 모델과 애플리케이션 정보 모델을 표현한 온톨로지를 사용하여 표현하는데, 컴포넌트 조립 시 SCDT와 매칭되는 비즈니스 엔티티 자동 검색, 데이터 타입 변환 커넥터 코드 자동 생성에 사용된다.

### 4.3 온톨로지 기반 DTT 속성 메타데이터

데이터 타입 컨버터 자동 생성을 위해 서비스 컴포넌트의 SCDT와 비즈니스 엔티티들의 속성에 부가하는 DTT 속성 메타데이터는 그림 4와 같이 정의하였다. 메타데이터는 속성의 개념(의미)을 기술하는 온톨로지의 *Class URI* 와 속성이 취할 수 있는 값을 표현한 *Semantic Value Type*, 타입 변환 코드 생성시 사용되는 *Converting Criteria* 로 구성된다.

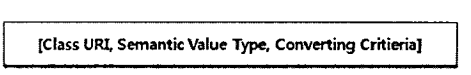


그림 4 DTT 속성 메타데이터 구조

**Class URI**

DTT 속성 메타데이터의 첫번째 구성요소는 두번째 구성요소가 *ClassName* 이나 *SubClassName* 일 때는 속성자체의 개념을 나타내는 온톨로지 클래스의 URI이고, 두번째 구성요소가 *SemanticProperty* URI 일 때는 속성이 속한 비즈니스 엔티티의 개념을 나타내는 온톨로지 클래스의 URI이다.

**Semantic Value Type**

두번째 구성요소는 비즈니스 엔티티의 속성이 취할 수 있는 의미론적 값의 타입으로 *'ClassName'*, *'SubClassName'*, *SemanticProperty* URI 중 하나가 된다. *'ClassName'* 은 비즈니스 엔티티 속성의 값으로 온톨로지 클래스 이름 자체를, *'SubClassName'* 은 첫번째 구성요소인 온톨로지 클래스가 Value Partition(코드 체계를 표현하기 위한 온톨로지 디자인 패턴)일 때 그것의 서브클래스 이름을 취할 수 있음을 나타낸다. *SemanticProperty* URI는 첫번째 구성요소인 온톨로지 클래스의 속성 중 SCDT/비즈니스 엔티티 속성의 의미를 나타내는 온톨로지 속성 URI를 메타데이터로 취하는 것을 표현한 것이다.

**Converting Critieria**

세번째 구성요소인 *Converting Critieria* 는 비즈니스 엔티티의 속성을 SCDT의 속성으로 변환하는 코드를 생성할 때 사용되는 변환 기준(converting critieria)이다. 표 1은 비즈니스 엔티티 속성의 데이터 타입과 *Semantic Value Type* 에 따라 *Converting Critieria* 가 취할 수 있는 종류를 분류한 것이다.

### 4.4 데이터 타입 컨버터 생성 알고리즘

데이터 타입 컨버터는 서비스 컴포넌트가 비즈니스 엔티티 대신 SCDT를 액세스할 때 비즈니스 엔티티 객체의 속성을 그에 대응하는 SCDT 속성으로 변환해주는 Get/Set 액세서 형식의 타입 변환 메소드들로 구성된다. 그림 5는 데이터 타입 컨버터 생성 과정 흐름도로서 온톨로지 및 메타데이터 정보들과 관련 지어 알고리즘 각 단계를 설명한다.

[P1] 서비스 데이터 컴포넌트와 비즈니스 엔티티 조립 시, 서비스 데이터 컴포넌트의 내장 데이터 타입의 속성들에 부가된 메타데이터와 비즈니스 엔티티의 속성들에 부가된 메타데이터들을 읽는다.

[P2] 각 속성의 의미에 해당하는 개념들을 도메인 온톨로지와 확장 온톨로지에서 추론 기능을 사용해 비교함으로써 매핑되는 속성들의 쌍들을 식별한다.

[P3] 식별된 모든 속성쌍들 중 변환 코드가 생성되지 않은 속성쌍을 선택한다.

[P4] 선택된 속성쌍의 메타데이터에서 속성 변환 기준을 읽어 그 변환 기준에 따라 분기한다.

[P5] [P4]에서 읽은 속성 변환 기준이 '없음(None)'면 두 속성의 데이터 타입을 비교하여 단순 데이터 타입 변환 코드를 생성한다.

[P6] [P4]에서 읽은 속성 변환 기준이 단위의 이름이면 두 속성의 값을 해당 단위에 맞게 변환하는 코드를 생성한다.

[P7] [P4]에서 읽은 속성 변환 기준이 온톨로지에서 범위나 값을 나타내는 개념이고, 속성의 의미가 밸류 세트 패턴의 휘처 개념이면서 속성이 취할 값이 개체 이름일 경우, 변환할 속성의 온톨로지와 변환될 속성의 온톨로지에서 밸류 세트의 각 개체들과 관계로 연결된 범위나 값들을 상호 비교하여 의미상 등가의 개체들을 찾고, 변환할 속성의 값에 해당하는 개체의 이름을 변환될 속성의 온톨로지에 있는 등가의 개체 이름으로 변환해주는 코드를 생성한다.

[P8] 모든 속성쌍에 대하여 변환 코드가 만들어질 때까지 [P3]에서 [P7] 단계를 반복하여 데이터 타입 컨버터 코드를 완성한다.

## 5. Case Study : 성적관리시스템

본 연구팀은 본 논문에서 제시한 온톨로지 기반 데이터 가변성 처리 기법 유효성을 검증하기 위해, 4.3절에서 설명한 컴포넌트의 인터페이스와 비즈니스 엔티티에 온톨로지를 적용한 메타데이터를 부가할 수 있게 지원

표 1 Converting Critieria 분류

| Class URI | Semantic Value Type | Property Data Type | Converting Criteria |
|---|---|---|---|
| 속성 자체의 의미를 나타내는 온톨로지 개념의 URI | ClassName | String | None |
| | | | 변환기준개념의 URI |
| | SubClassName | String | None |
| | | | 변환기준개념의 URI |
| 속성이 속한 내장데이터 타입이나 비즈니스 엔티티의 의미를 나타내는 URI | SemanticProperty URI | Numeric | None |
| | | | Unit Name |
| | | String | None |
| | | | Format String |
| | | Date | Format String |

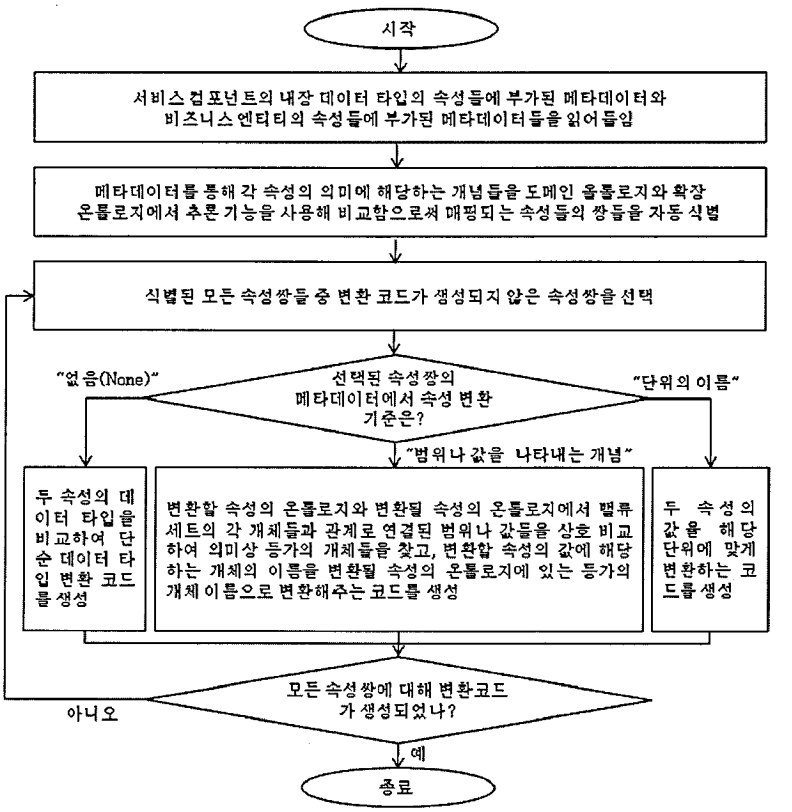


그림 5 데이터 타입 컨버터 자동 생성 알고리즘

하는 컴포넌트 개발 도구와 4.4절에서 설명한 데이터 타입 컨버터 생성 알고리즘에 따라 컨버터를 자동 생성하는 조립 도구를 구현하였다. 본 장에서는 그 개발 도구와 조립 도구에 가상의 성적관리시스템 개발을 적용한 구현 예를 통해, 4장에서 제안한 온톨로지 기반 DTT 컴포넌트 모델에서 데이터 타입 컨버터가 생성되는 방법에 대해 구체적으로 설명한다. 본 장에서 다루게 될 예는 Seoul National University(앞으로 SNU라 표기)와 학점 교류 협약을 맺은 Ewha University(앞으로 Ewha라 표기)의 학점 관리 프로그램을 온톨로지 기반 DTT 컴포넌트 모델에 따라 개발한 것이다. 이 프로그램은 Ewha 소속의 특정 학생이 취득한 GPA를 계산할 때 이 학생이 SNU에서 수강한 과목이 있다면, SNU에게 학생이 수강한 과목들의 학점을 요구하고, 이를 Ewha 학점체계로 전환한 다음 Ewha에서 수강한 과목의 학점들과 함께 계산하여 GPA를 구한다. 시스템을 개발할 당시 Ewha는 SNU와 학점 교류 협약을 맺었으나, 점차 학점 교류할 대학을 추가할 계획이 있다고 가정한다. 먼저 도메인 엔지니어링을 통해 서비스 요구사항과 가변성을 식별한 후, 컴포넌트와 비즈니스 엔티티를 식별하고 데이터 가변성을 효율적으로 처리하기 위한 성적관리 시스템 컴포넌트 아키텍처를 그림 6과 같이 설계하였다.

**서비스 요구사항**

- 학생 아이디와 년도, 학기를 매개변수로 받아 학생의 해당 년도/학기의 학점을 메시지로 전송하는 기능을 제공해야 한다.
- 학점 교류 협약을 맺은 대학에서 수강한 과목은 소속 대학의 학점 체계로 변환한 후 평점을 계산한다.

**가변 요구사항**

- 학점 체계를 미리 알 수 없는 다양한 대학들과 학점 교류 협약이 늘어나도 평점을 계산하는 서비스 컴포넌트는 다른 대학의 학점 체계를 소속 대학의 학점 체계로 자동 변환하여 GPA가 계산할 수 있어야 한다.

### 5.1 성적관리 시스템 온톨로지 모델 및 컴포넌트 아키텍처

그림 7은 각 학교별 성적관리 애플리케이션 온톨로지 중 본 예에서 필요한 부분을 나타낸 것이다. Ewha는 그림 7의 (1)과 같이 성적관리 도메인 온톨로지를 확장하여 Ewha 온톨로지를 구축하였다. 성적관리 도메인

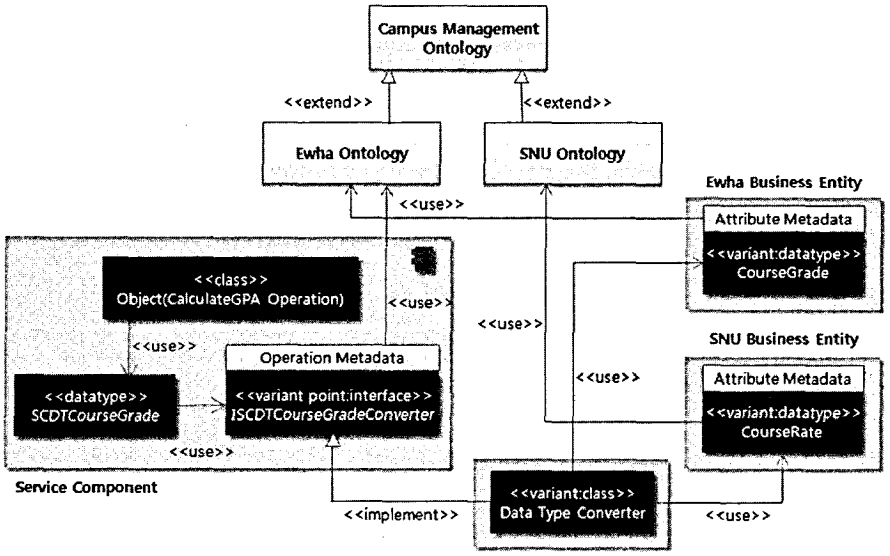


그림 6 온톨로지 기반 데이터 가변성 처리 기법을 적용한 성적관리 시스템 컴포넌트 아키텍처

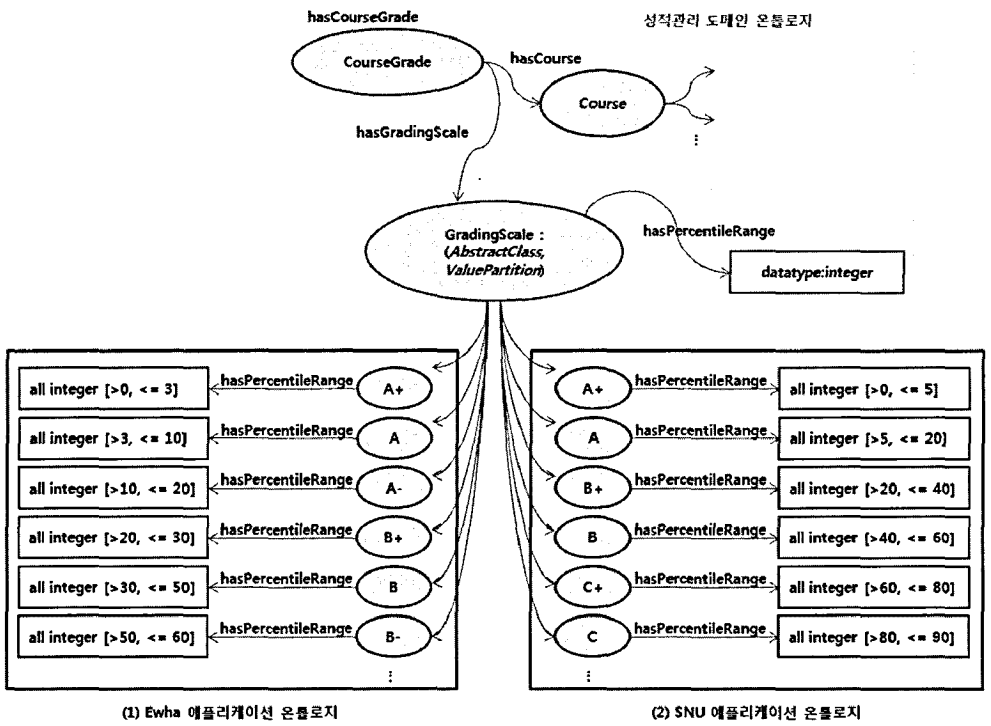


그림 7 성적관리 온톨로지(partial)

온톨로지는 교육과학기술부(Ministry of Education)에서 대학간 학점 교류를 장려하기 위해 구축한 온톨로지로 가정한다. 도메인 온톨로지에서 GradingScale이라는 클래스는 각 학교마다 달라지는 성적 체계로 확장할 수 있도록 AbstractClass와 ValuePartition으로 설정하였다. AbstractClass는 GradingScale의 서브 클래스를 각 대학이 도메인 온톨로지를 확장하여 애플리케이션 온톨로지를 생성해야 하는 책임이 있음을 의미한다. 또한

ValuePartition으로 정의함으로써 각 사이트는 GradingScale의 코드 체계를 서브클래스로 표현한다. 그림 7의 (1)은 Ewha가 도메인 온톨로지에 있는 GradingScale의 서브클래스로 A+, A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F 클래스를 갖도록 확장한 것을 보여준다. 그리고 성적을 나타내는 각 서브클래스마다 학생의 비율을 나타내는 Range 속성 hasPecentileRange의 값을 갖는다. 그림 7의 (2)는 SNU의 애플리케이션 온톨로지 일부로 A+, A, B+, B, C+, C, D+, D, F 로 구성된 학점 체계이다.

### 5.2 Course Grade 비즈니스 엔티티

그림 8은 Ewha 성적관리 애플리케이션에서 사용하는 비즈니스 엔티티인 CourseGrade 클래스의 c# 코드를 보여준다. 각 속성에 커스텀 애트리뷰트 형식으로 부가된 DTT 속성 메타데이터에는 5.1절의 온톨로지를 사용하여 속성의 의미와 변환기준이 기술되어 있다. 이를 상세히 설명하면 비즈니스 엔티티인 CourseGrade에 DTT 속성 메타데이터에서 사용되는 온톨로지들의 네임스페이스가 MetaOntologyNS라는 커스텀 애트리뷰트로 부가되어 있다. 도메인 온톨로지의 네임스페이스는 CampusMgt로, Ewha 애플리케이션 온톨로지의 네임스페이스를 Ewha로 정의하고 있음을 볼 수 있다. CourseGrade의 각 속성에는 DTT 속성 메타데이터가 DTTPropertyMetaTriple이라는 커스텀 애트리뷰트로 부가되어 있다. CourseGrade의 속성 중 하나인 Grade에 부가된 DTT 속성 메타데이터의 첫번째 요소는 "CampusMgt:GradingScale"로 Grade 속성이 도메인 온톨로지에 정의되어 있는 GradingScale이라는 클래스의 개념(의미)을 갖는 것을 기술한다. 두번째 구성요소는 "Ewha:SubClassName"으로 Ewha 애플리케이션 온톨로지에서 확장한 GradingScale의 서브클래스 중 하나의 이름을 Grade 속성의 값으로 취할 수 있음을 나타낸다. 세번째 구성요소는 "CampusMgt:hasPecentileRange"로 비즈니스 엔티티 CourseGrade의 속성인 Grade를 SCDT의 속성으로 변환할 때, 해당 속성들에 적용된 온톨로지의 Range 속성 hasPecentileRange의 값이 변환 기준이 됨을 의미한다.

그림 9는 SNU 성적관리 애플리케이션에서 사용하는 비즈니스 엔티티인 CourseRate 클래스의 코드이다. CourseRate 비즈니스 엔티티에는 StudentId 등 6개의 속성이 선언되어 있고, 각각의 속성에 DTT 속성 메타데이터에 부가되어 그 의미와 변환 기준을 표현하고 있다.

### 5.3 GPA 계산 서비스 컴포넌트

그림 10은 Ewha 성적관리 시스템의 서비스 컴포넌트에서 비즈니스 엔티티를 대신하여 사용될 SCDT 클래스 SCDTCourseGrade를 보여준다. SCDT에는 StudentId,

```
[MetaOntologyNS(OntologyURI =
   "http://www.mest.go.kr/CampusManagement/CoreCampusManagement.owl",
   QNamePrefix = "CampusMgt")]
[MetaOntologyNS(OntologyURI =
   "http://www.ewha.ac.kr/CampusManagement/EwhaCampusManagement.owl",
   QNamePrefix = "Ewha")]
public class CourseGrade : Entity
{
   [DTTPropertyMetaTriple("CampusMgt:Student",
                  "CampusMgt:hasStudentId",
                  "None")]
   public string StudentId { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Course",
                  "CampusMgt:hasTitle",
                  "None")]
   public string CourseTitle { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Course",
                  "CampusMgt:hasYear",
                  "None")]
   public int Year { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Course",
                  "CampusMgt:hasSemester",
                  "None")]
   public int Semester { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:GradingScale",
                  "Ewha:SubClassName",
                  "CampusMgt:hasPercentileRange")]
   public string Grade { get; set; }
}
```

그림 8 Ewha 비즈니스 엔티티 코드 예

```
[MetaOntologyNS(OntologyURI =
   "http://www.mest.go.kr/CampusManagement/CoreCampusManagement.owl",
   QNamePrefix = "CampusMgt")]
[MetaOntologyNS(OntologyURI =
   "http://www.snu.ac.kr/CampusManagement/SNUCampusManagement.owl",
   QNamePrefix = "SNU")]
public class CourseRate : Entity
{
   [DTTPropertyMetaTriple("CampusMgt:Student",
                  "CampusMgt:hasStudentId",
                  "None")]
   public string LearnerId { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Course",
                  "CampusMgt:hasTitle",
                  "None")]
   public string CourseName { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Course",
                  "CampusMgt:hasYear",
                  "None")]
   public int Year { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Course",
                  "CampusMgt:hasSemester",
                  "None")]
   public int Semester { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:GradingScale",
                  "SNU:SubClassName",
                  "CampusMgt:hasPercentileRange")]
   public string Rate { get; set; }

   [DTTPropertyMetaTriple("CampusMgt:Instructor",
                  "CampusMgt:hasId",
                  "None")]
   public string InstructorId { get; set; }
}
```

그림 9 SNU 비즈니스 엔티티 코드 예

Grade 속성을 갖는다. StudentId에 부가된 DTT 속성 메타데이터의 첫번째 요소는 "CampusMgt:Student"로 StudentId 속성이 도메인 온톨로지에 정의되어 있는 Student이라는 클래스의 개념(의미)을 갖는 것을 기술한다. 두번째 구성요소는 "CampusMgt:hasStudentId"로 속성값을 표현하고 있다. Grade 속성은 메타데이터의 첫번째 구성요소는 "CampusMgt:GradingScale"로

Grade 속성이 도메인 온톨로지에 정의되어 있는 GradingScale이라는 클래스의 개념(의미)을 갖는 것을 기술한다. 두번째 구성요소는 "Ewha:SubClassName"으로 Ewha 애플리케이션 온톨로지에서 확장한 GradingScale의 서브클래스 중 하나의 이름을 Grade 속성의 값으로 취할 수 있음을 나타낸다. 세번째 구성요소는 "CampusMgt:hasPecentileRange"로 비즈니스 엔티티와 변환할 때 적용될 변환 기준을 나타낸다. 또한 SCDT 클래스 SCDTCourseGrade에는 서비스 컴포넌트의 GPA 계산 메소드가 매개 변수로 넘겨 받은 비즈니스 엔티티를 SCDT인 SCDTCourseGrade의 객체로 타입 캐스팅할 때 사용되는 변환 오퍼레이터와 이 오퍼레이터에서 사용할 생성자가 정의되어 있다.

그림 11은 서비스 컴포넌트에 정의된 데이터 타입 컨버터 인터페이스 ISCDTCourseGradeConverter 코드를 보여준다. 컨버터 인터페이스는 비즈니스 엔티티 속성과 SCDT의 속성간 변환을 담당하는 메소드들로 구성된다.

그림 12는 서비스 컴포넌트의 CalculateGPA 메소드 코드를 보여준다. 이 메소드는 학생 아이디, 수강연도, 수강학기를 입력받아 GPA를 계산하여 결과로 내준다. CalculateGPA 메소드는 먼저 학생이 해당 학기에 수강한 모든 과목의 성적을 리스트로 돌려주는 하위 컴포넌트의 메소드를 호출한다. 하위 컴포넌트의 메소드 GetCourceOfStudent는 소속 대학인 Ewha의 비즈니스

```
[ClassType(ClassType.SelfDefinedTypeClass)]
[MetaOntologyNS(OntologyURI =
    "http://www.mest.go.kr/CampusManagement/CoreCampusManagement.owl",
    QNamePrefix = "CampusMgt")]
[MetaOntologyNS(OntologyURI =
    "http://www.ewha.ac.kr/CampusManagement/EwhaCampusManagement.owl",
    QNamePrefix = "Ewha")]
public class SCDTCourseGrade : Entity
{
    [DTTPropertyMetaTriple("CampusMgt:Student",
                           "CampusMgt:hasStudentId",
                           "None")]
    public string StudentId
    {
        get { return TypeConverter.GetStudentId(entity); }
        set { TypeConverter.SetStudentId(entity, value); }
    }

    [DTTPropertyMetaTriple("CampusMgt:GradingScale",
                           "Ewha:SubClassName",
                           "CampusMgt:hasPercentileRange")]
    public string Grade
    {
        get { return TypeConverter.GetGrade(entity); }
        set { TypeConverter.SetGrade(entity, value); }
    }

    [DependencyInjection()]
    public static ISCDTCourseGradeConverter TypeConverter;
    private Entity entity;

    public SCDTCourseGrade(Entity entity)
    {
        this.entity = entity;
    }

    public static explicit operator SCDTCourseGrade(Entity entity)
    {
        return new SCDTCourseGrade(entity);
    }
}
```

그림 10 SCDT 코드 예

```
public interface ISCDTCourseGradeConverter
{
    string GetStudentId(Entity entity);

    void SetStudentId(Entity entity, string newValue);

    string GetGrade(Entity entity);

    void SetGrade(Entity entity, string newValue);
}
```

그림 11 DTT 인터페이스 코드 예

```
[ServiceMethod]
[RequiredMethod("IRqCourseMgt", "GetCoursesOfStudent", 1)]
[Parameter("id", "CampusMgt:Student", "CampusMgt:hasId", "None")]
[Parameter("year", "CampusMgt:Course", "CampusMgt:hasYear", "None")]
[Parameter("semester", "CampusMgt:Course", "CampusMgt:hasSemester", "None")]
[Return("CampusMgt:Course", "CampusMgt:hasGPA", "None")]
public float CalculateGPA(string id, int year, int semester)
{
    List<Entity> courseList =
        RIRqCourseMgt.GetCoursesOfStudent(id, year, semester);

    float grade = 0;

    int cnt = courseList.Count;
    float total = 0;

    foreach (Entity obj in courseList)
    {
        SCDTCourseGrade course = (SCDTCourseGrade)obj;
        switch (course.Grade)
        {
            case "A+": total += 4.3f; break;
            case "A": total += 4.0f; break;
            case "A-": total += 3.7f; break;
            case "B+": total += 3.3f; break;
            case "B": total += 3.0f; break;
        }
    }

    grade = total / cnt;

    return grade;
}
```

그림 12 서비스 컴포넌트 코드 예

엔티티 CourseGrade 객체들과 더불어 학점을 교류하는 모든 타대학에서 해당 학생이 수강한 과목들의 성적을 담고 있는 객체들을 리턴해야 하므로, 모든 비즈니스 엔티티가 상속하는 루트 데이터 타입인 Entity 타입의 리스트를 사용한다. 서비스 컴포넌트의 메소드 CacluateGPA는 하위 컴포넌트로부터 받은 리스트로부터 Entity를 하나씩 읽어 이를 SCDT인 SCDTCourseGrade타입의 객체로 타입 캐스팅한다. 이후 SCDT 객체들의 속성들을 액세스해서 GPA를 계산하여 리턴한다. CacluateGPA 메소드가 SCDT 객체의 속성 Grade를 액세스할 때마다 Grade의 Get 액세서가 호출되고, Get 액세서는 데이터 타입 컨버터 객체의 GetGrade 메소드를 호출하면서 실제의 비즈니스 엔티티 객체의 레퍼런스를 매개변수로 전달함으로써 학생이 각 대학에서 취득한 성적을 소속 대학의 성적으로 변환된 값을 얻게 된다.

### 5.4 데이터 타입 컨버터 컴포넌트

그림 13은 비즈니스 컴포넌트들과 서비스 컴포넌트들을 조립할 때 생성되는 데이터 타입 컨버터 SCDTCourseGradeConverter의 코드를 보여준다. 애플리케이션을 개발할 당시 Ewha 대학은 SNU와 학점 교류 협약을 맺고 있으므로 SCDTCourseGradeConverter는

```
[ClassType(ClassType.EntityConverterClass)]
[TargetEntity("EwhaBE.dll", "EwhaBE.CourseGrade")]
[TargetEntity("SnuBE.dll", "SnuBE.CourseRate")]
public class SCDTCourseRateConverter : ClassLibrary1.ISCDTCourseRateConverter
{
    . . .

    [TargetProperty("Grade")]
    public string GetGrade(Entity entity)
    {
        string rVal = "";
        if(entity is EwhaBE.CourseGrade)
        {
            EwhaBE.CourseGrade obj = (EwhaBE.CourseGrade)entity;
            rVal = obj.Grade;
        }
        else if(entity is SnuBE.CourseRate)
        {
            SnuBE.CourseRate obj = (SnuBE.CourseRate)entity;
            rVal = ConvertSNURate2SCDTGrade(obj.Rate);
        }

        return rVal;
    }

    private string ConvertSNURate2SCDTGrade(string rate)
    {
        string rVal = rate;

        switch(rate)
        {
            case "A+": rVal = "A+"; break;
            case "A": rVal = "A-"; break;
            case "B+": rVal = "B"; break;
            case "B": rVal = "B-"; break;
            case "C+": rVal = "C+"; break;
            case "C": rVal = "C"; break;
            case "D+": rVal = "D+"; break;
            case "D": rVal = "D-"; break;
            case "F": rVal = "F"; break;
        }

        return rVal;
    }

    . . .
}
```

그림 13 Ewha CourseGrade와 SCDT간 데이터 타입 컨버터 구현 코드 예

Ewha의 비즈니스 엔티티인 CourseGrade 객체와 SNU의 비즈니스 엔티티인 CourseRate 객체의 속성들을 서비스 컴포넌트의 SCDT인 SCDTCourseGrade 객체 속성들로 변환하는 메소드들이 구현되어 있다. SNU Couse-Rate 객체의 Rate 값을 SCDTCourseGrade 객체 의 Grade 값으로 변환해주는 유틸리티 메소드 Convert-SNURate2SCDTGrade는 각 속성에 부가된 메타데이터의 온톨로지 정보(그림 7)를 참조하여 자동 생성된 것이다. 이 코드들은 4.4절에서 설명한 데이터 타입 컨버터 자동 생성 알고리즘에 의해 생성되므로 추후 학점을 교류하는 대학이 늘어나더라도 자동으로 재생성할 수 있다.

### 5.5 고찰(Discussion)

본 논문에서 제안한 온톨로지 기반 DTT 컴포넌트 모델은 선행 연구에서 제안한 DTT 컴포넌트 모델의 데이터 가변성 처리 효율을 더욱 높이기 위한 것으로, 기존 DTT 컴포넌트 모델에서 데이터가 변할 때 서비스 컴포넌트들을 수정 없이 재사용하기 위해 데이터 타입 컨버터를 수작업으로 개발하는 부담을 없애기 위한 것이다. 표 2에서 두 모델의 차이점을 개발 단계와 조립 단계로 구분하여 비교하였다.

표 2 DTT 컴포넌트 모델과 온톨로지 기반 DTT 컴포넌트 모델 비교

| | DTT 컴포넌트 모델 | 온톨로지 기반 DTT 컴포넌트 모델 |
|---|---|---|
| 컴포넌트 개발단계 | ·인터페이스/속성 메타데이터에 자연어 기반 설명 부가<br>·매뉴얼, 샘플 프로그램 작성 | ·도메인/애플리케이션 온톨로지 구축<br>·인터페이스/속성에 온톨로지 기반 메타데이터 부가 |
| 컴포넌트 조립단계 | ·개발자가 인터페이스 속성에 부가된 자연어 기반 메타데이터 의미 해석<br>·메뉴얼과 샘플 프로그램을 참조하여 데이터 타입 컨버터 코드를 개발자가 작성 | ·조립도구가 온톨로지 기반 메타데이터를 해석하여 데이터 타입 컨버터 코드 자동 생성 |

DTT 컴포넌트 모델에서는, 서비스 컴포넌트를 개발할 때 데이터 가변성 처리에서 핵심적인 역할을 하는 데이터 타입 컨버터 코드를 에러 없이 개발할 수 있도록 하기 위해, 각 인터페이스에 대한 정확하고 자세한 개발자 매뉴얼과 사용 예에 대한 샘플 코드 등을 작성해야 한다. 반면 온톨로지 기반 DTT 컴포넌트 모델에서는 매뉴얼 이나 샘플 코드를 작성할 필요 없이 각 인터페이스 오퍼레이션들의 매개 변수들에 의미론적으로 매핑되는 온톨로지 모델의 개념이나 속성을 선택해서 메타데이터로 부여한다. 컴포넌트 조립 단계에서는 DTT 컴포넌트 모델은 개발자가 매뉴얼이나 샘플 코드를 이해하고 에러없이 데이터 타입 컨버터를 개발하는 수작업이 필요하지만, 온톨로지 기반 DTT 컴포넌트 모델에서는 조립도구가 자동으로 데이터 타입 컨버터를 생성해준다.

온톨로지 기반 DTT 컴포넌트 모델이 제대로 작동하기 위해서는, 일반 개발자들에게 다소 생소한 온톨로지 언어로 도메인이나 애플리케이션의 정보 모델을 작성해야 하는 부담이 따른다. 하지만 이런 부담은 정보 모델을 반 형식적인(semi-formal) UML이나 비형식적인(informal) 자연어를 사용해 표현했을 때 사용자, 아키텍트, 개발자 간 오해로 인해 빈번하게 발생하는 재개발이나 디버깅 부담에 비교하면 충분히 감내할 수 있는 부담이다. 또 온톨로지 모델 설계를 돕기 위해, 일반 개발자들에게 익숙한 클래스 다이어그램 편집기와 유사한 그래픽 도구와 정보 모델을 정밀하게 표현할 수 있게 하는 공리들을 생성해주는 마법사 등을 만들어 보급하면 그 부담을 크게 경감시킬 수 있다.

온톨로지 기반 데이터 가변성 처리 기법에서 사용한 컨버터 자동 생성 방법은 서비스 컴포넌트들을 조립할 때 필요한 인터페이스 중재 코드 생성에도 똑같이 적용

할 수 있다. 데이터 타입 컨버터 자동 생성과 인터페이스 중재 코드 자동 생성은 컴포넌트 조립 시 발생하는 컴포넌트 적응(adaptation) 비용을 거의 제로(0) 수준으로 만들어 컴포넌트 재사용성을 높이고 부담 없는 조립 변경을 가능케 한다. 이로 인해 자유스러운 컴포넌트 아키텍처 변경이 가능하게 되어, 데이터 가변성뿐만 아니라 기능 가변성이나 품질 가변성 처리도 쉽게 수용할 수 있게 된다. 또한 컴포넌트들을 조립한 그래픽 모델만 완성하면 컴포넌트들이 상호 작용하기 위한 커넥터 코드들을 자동 생성하는 조립 도구는 조립 개발자들에게 "조립 모델이 곧 애플리케이션"인 추상성을 제공하므로 소프트웨어 생산성 제고에 크게 기여할 것으로 기대된다.

## 6. 결 론

다계층 구조의 기업 애플리케이션에서 비즈니스 엔티티의 변화는 비즈니스 엔티티 자체의 수정뿐 아니라 이를 다루는 각 계층의 서비스 컴포넌트 모두를 수정해야 하는 부담이 따른다. 본 연구팀이 이전 연구에서 제안했던 DTT 컴포넌트 모델은 서비스 컴포넌트들과 비즈니스 엔티티들 간의 직접적인 결합을 없앰으로써 서비스 컴포넌트들을 수정하지 않고도 새로운 비즈니스 엔티티들을 처리할 수 있게 한 반면, 이들을 중재하는 데이터 타입 컨버터를 개발해야 하는 부담을 야기한다. 속성간 데이터 타입 변환이나 단위 변환 등만이 필요한 보통의 경우에 데이터 타입 컨버터 생성이 서비스 컴포넌트 수정에 비해 훨씬 효율적이긴 하지만, 코드 체계가 서로 다른 속성 값을 갖는 다양한 비즈니스 엔티티들을 동시에 처리해야 하는 경우처럼 복잡한 데이터 가변성을 처리하는 데이터 타입 컨버터를 에러 없이 생성하는 일은 쉽지 않다.

본 논문에서는 서비스 컴포넌트의 SCDT와 비즈니스 엔티티의 각 속성에 대한 메타데이터로 온톨로지를 사용함으로써 데이터 타입 컨버터를 에러 없이 자동 생성할 수 있는 온톨로지 기반 DTT 컴포넌트 모델을 제안하였다. DTT 속성 메타데이터의 구조와 해석 방법을 구현 레벨에서 정의하여, 기업 애플리케이션 개발에서 일반적으로 사용되는 닷넷이나 자바 기술 등에 적용하여 실현할 수 있게 하였다. 또 서로 다른 학점 체계를 갖는 다양한 대학들과 학점교류를 하면서 평점 계산을 수행하는 구체적인 예를 통해 온톨로지 기반 DTT 컴포넌트 모델을 자세히 설명하고 실효성을 검증하였다.

온톨로지 기반 DTT 컴포넌트 모델을 사용하면 기업 애플리케이션의 데이터 가변성을 매우 효율적으로 처리할 수 있지만, 이러한 효율성은 잘 설계된 온톨로지를 전제로 한다. 소프트웨어 개발에 관여하는 설계자나 개발자들은 보통 온톨로지에 익숙하지 않기 때문에 이들이 정확하고 쉽게 온톨로지를 만들 수 있게 해주는 모델링 도구가 필요하다. DTT 속성 메타데이터는 매치되는 속성 식별, 데이터 타입 변환, 단위 변환, 포맷 변환, 코드 체계 변환 등 SCDT와 비즈니스 엔티티의 1:1 속성변환을 지원한다. 향후 보다 더 복잡한 시스템에서 활용하기 위해 m:n 속성 변환 기술에 대한 연구가 필요하다.

## 참 고 문 헌


[1] Yoonsun Lim, Myung Kim, Seongnam Jeong and Anmo Jeong, "Data Type-Tolerant Component Model: A Method to Process Variability of Externalized Data," *Journal of KIISE: Software and Applications*, vol.36, no.5, pp.386-396, May 2008. (in Korean)

[2] John Cheesman and John Daniels, "UML Components: A Simple Process for Specifying Component-Based Software," *Addison Wesley*, 2001.

[3] "Patterns & Practices, Application Architecture for .NET: Designing Applications and Services," *Microsoft Corporation*, 2002.

[4] Martin Fowler, "Patterns of Enterprise Application Architecture," *Addison Wesley*, 2003.

[5] Eric Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software," *Addison Wesley*, 2003.

[6] Barry Keepence and Mike Mannion, "Using patterns to model variability in product families," *IEEE Software*, vol.16, issue: 4, pp.102-108, 1999.

[7] Matthias Clauß, "Generic Modeling using UML extensions for variability," *OOPSLA 2001, Workshop on Domain Specific Visual Languages*, 2001.

[8] Hassan Gomaa and Diana L Webber, "Modeling adaptive and evolvable software product lines using the variation point model," *Proceedings of the 37th Annual Hawaii International Conference.*

[9] Joseph W. Yoder, Federico Balaquer and Ralph Johnson, "Architecture and Design of Adaptive Object-Models," *Intriguing Technology from OOPSLA 2001, ACM SIGPLAN Notices*, vol.36, pp.50-60, ACM Press, December 2001.

[10] Joseph W. Yoder, "Adaptive Object-Model Architecture: How to Build Systems That Can Dynamically Adapt to Changing Requirements," in *Tutorials, 19th ECOOP*, Jul. 2005.

[11] OWL-S 1.0 release [online] Available: http://www.daml.org/services/ owl-s/1.0/.

[12] Keita Fujii and Tatsuya Suda, "Dynamic Service Composition Using Semantic Information," in *Proc. 2nd Int. Conf. Service Oriented Comput.*, Nov. 2004.

[13] Ponnekanti, S. R. and Fox, A., "SWORD: A Developer Toolkit or Web Service Composition," *to appear in 11th World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, May 7-11, 2002.

임 윤 선

1987년 중앙대학교 수학과 학사. 2003년 이화여자대학교 컴퓨터학과 석사. 2004년~현재 이화여자대학교 컴퓨터학과 박사 과정. 관심분야는 온톨로지, 소프트웨어 재사용, 프로덕트라인 공학, 지식공학 등

김 명

1981년 이화여자대학교 수학과 학사. 1983년 서울대학교 계산통계학과 석사. 1993년 캘리포니아 주립대학교(산타바바라) 컴퓨터학과 박사. 1993년~1994년 캘리포니아 주립대학교(산타바바라) 컴퓨터학과 Postdoc, 강사. 1995년~현재 이화여자대학교 컴퓨터학과 교수. 관심분야는 온톨로지, 고성능 컴퓨팅, 소프트웨어 재사용, 지식공학, 스트림 데이터처리 등