EPCglobal TDT 1.0 표준을 따르는 태그 데이타 변환기의 설계와 구현

(Design and Implementation of an EPCglobal TDT 1.0 Compliant Tag Data Translator)

김 성 진 † 송 하 주 ††
(Sung-Jin Kim) (Ha-Joo Song)

요 약 EPCglobal TDT은 RFID 태그 데이타의 레벨(포맷)과 이들 간의 상호 변환에 대한 표준을 제시하고 있다. 이것은 태그 데이타 변환이라고 하며 이백 가지가 넘는 조합이 가능하다. 본 논문에서는 태그 데이타 변환 프로그램을 개발함에 있어 TDT XML 명세로부터 소스 코드를 자동으로 생성하기 위한 코드 생성기를 제안한다. 코드 생성기를 이용하면 변환 코드를 간단하게 작성할 수 있으며 추후 TDT 명세가 변경되더라도 유지 보수를 신속하게 할 수 있다.

키워드 : RFID, EPCglobal, 태그 데이타 변환, 코드 생성

Abstract EPCglobal TDT specifies standards on RFID tag representations and conversion rules among them. According to the rule, more than 200 combinations of code conversions are possible. In this paper, we propose a code generation scheme for developing TDT code conversion program, that imports TDT XML specification and generates conversion codes. The proposed scheme makes it easy to develop and to maintain the conversion codes.

· 이 논문은 2008년 교육과학기술부로부터 지원받아 수행된 연구임(지역거점연구단육성사업/차세대물류IT기술연구사업단)
· 이 논문은 제34회 추계학술대회에서 'EPCglobal TDT 1.0 표준을 따르는 태그 데이타 변환기의 설계와 구현'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 부경대학교 전자컴퓨터정보통신공학부
manghon2@hanmail.net
†† 정 회 원 : 부경대학교 전자컴퓨터정보통신공학부 교수
hajusong@pknu.ac.kr

논문접수 : 2007년 12월 7일
심사완료 : 2008년 10월 20일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제9호(2008.12)

Key words : RFID, EPCglobal tag data translation, code generation

1. 서 론

EPC(Electronic Product Code)[1]는 RFID 태그의 식별을 위한 코드 체계이며 EPC Tag Data Standard (TDS)[2] 표준은 기존의 코드 체계(scheme)를 EPC로 표시하기 위한 명세를 정의하고 있다. EPC Tag Data Translation (TDT)[3] 표준은 EPC 코드의 구조를 컴퓨터로 처리하기에 용이하도록 XML을 이용하여 명세를 나타내고 있는데 다음과 같은 인터페이스를 사용하여 레벨(level, 태그 포맷)들 간의 태그 데이타 변환 규칙을 정의하고 있다.

```
public String translate(
    String epcIdentifier,
    String parameterList,
    String outputFormat)
throws TDTTranslationException
```

여기서 epcIdentifier, parameterList, outputFormat은 각각 입력 코드, 옵션, 출력될 레벨을 의미하고 변환된 코드는 리턴값으로 전달된다. TDT 에서의 변환은 동일 코드 체계 내에서의 레벨 변환만을 지원하는데 BINARY, TAG_ENCODING, PURE_IDENTITY, LEGACY, ONS_HOSTNAME의 5가지 레벨을 사용할 수 있다. 태그 데이타 변환의 입력으로는 4종의 레벨을 사용할 수 있고 출력 레벨은 5종이 모두 가능하므로 각 코드 체계별로 가능한 태그데이타 변환의 조합은 20가지가 된다. 현재 EPCglobal의 Data Standards Version 1.0(TDT v1.0)에서 정의하고 있는 코드 체계는 모두 13가지(GID-96, SGTIN-64/96, SSCC-64/96, SGLN-64/96, GRAI-64/96, GIAI-64/96, USDOD-64/96)이다. 따라서 전체적으로는 260가지의 태그 데이타 변환 코드가 필요하다. 각 코드 체계별로 변환 코드를 수작업으로 작성하는 것은 개발 비용과 유지보수 측면에서 좋지 않다. 이에 본 논문에서는 TDT v1.0의 XML 명세를 입력하면 해당 변환 모듈 코드를 생성해 주는 코드 생성기(code generator)를 제안한다. 제안하는 코드 생성기는 13가지의 코드 체계별 변환 코드를 자동 생성한다. 따라서 변환 코드를 쉽게 개발할 수 있고 향후 새로운 코드 체계가 추가되거나 변환 절차가 변경되더라도 코드를 유지보수하기에 용이하다.

2. 관련 연구

TDT 기능은 리더, RFID 미들웨어[4], RFID 응용 프로그램 등 EPCglobal에서 제시하고 있는 EPC-Network의 어떠한 계층에서도 사용할 수 있다. TDT

구현의 대표적인 예로는 Accada[5], Avicon 등이 있다. Accada는 EPC Network의 표준 명세를 구현하기 위한 오픈 소스 프로그램으로 TDT v1.0 표준을 구현한 자바 라이브러리를 제공하고 있다. Accda의 경우, 코드 변환을 담당하는 TDT 엔진 모듈이 TDT XML 명세 파일을 실행시에 동적 파싱한 후 변환을 수행한다. Oracle은 데이타베이스 시스템 수준에서 TDT 기능을 지원한다. TDT 지원 방식을 변환 코드의 생성 시기에 따라 구분한다면 크게 동적변환 방식과 정적변환 방식으로 구분할 수 있다. 동적변환 방식은 실행시(runtime)에 TDT 명세 XML 파일를 읽고 변환 코드를 생성하여 수행하는 방식이며 Accada는 여기에 해당한다. 정적변환 방식은 미리 작성된 변환코드를 수행하는 것이다. 정적변환 방식은 한번 작성된 소스 코드를 컴파일한 후 계속해서 사용하는 것이므로 실행효율이 뛰어나지만 TDT 명세가 변경되는 경우 변환코드를 다시 작성해야하는 단점이 있다. 본 논문에서는 정적변환의 실행효율을 유지하면서도 변환코드의 유지보수를 쉽게 하기 위해 코드생성기를 사용하였다.

3. 태그 데이타 변환기의 설계

3.1 태그 데이타 변환기의 구성도

태그 데이타 변환기는 크게 코드 생성부와 변환 수행부로 나눌 수 있다(그림 1 참조). 변환 수행부는 XML 명세로부터 태그 데이타 변환 코드를 생성하는 모듈이다. 생성된 코드와 미리 작성된 코드를 합쳐서 라이브러리 형태로 만든 것이 변환 수행부이다. 따라서 실제 RFID 미들웨어 또는 응용프로그램에는 변환 수행부 라이브러리만 제공된다. 코드 생성부는 이 라이브러리를 재구축해야 할 경우에만 사용한다.

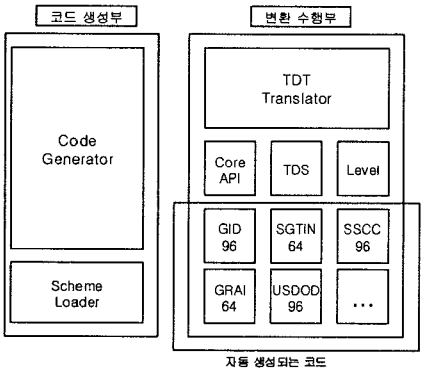

그림 1 태그 데이타 변환기의 구성도

3.2 코드 생성부

코드 생성부는 크게 보아 Scheme Loader클래스와 Code Generator 클래스로 구성된다. SchemeLoader 클래스는 입력된 TDT 정의 XML 파일에서 표현하는 코드 스킴의 구조를 자바 객체로 표현하는 역할을 한다. 이를 위해 Scheme, Level, Option, Rule, Field 라는 이름의 클래스들을 사용한다. Code Generator 클래스는 Scheme Loader가 전달해주는 내부적인 자료구조를 이용하여 변환 코드를 생성한다. 스킴별로 필드나 메소드를 추가하고, TDS와 Level 추상클래스의 메소드들을 구현하기 위한 메소드의 몸체(body)를 생성한다.

3.3 변환 수행부

변환 수행부는 변환을 위한 API를 제공하는 TDTTranslator 클래스와 변환 모듈 전체에서 사용되는 Core API, 13가지의 변환 모듈의 공통된 변환 규칙과 사용 자료들을 일반화한 TDS 클래스, Level 클래스, 그리고 Code Generator로부터 자동 생성된 13가지 변환 클래스로 구성된다. 변환 수행부는 미리 작성해야 할 부분과 Code Generator에 의해 자동으로 생성될 부분으로 구성된다. 미리 작성해야 할 부분은 TDTTranslator, TDS, Level, CoreAPI 그리고 TDTTranslationException의 하위 예외 클래스들로 구성된다. 자동으로 생성될 부분은 각 코드체계마다 생성되는 TDS의 구체화 클래스들이다. 변환 수행부는 각 코드 체계들의 태그 데이타 변환 과정 중에서 공통적으로 수행되는 부분을 일반화하여 TDS라는 추상클래스로 작성하고, 이를 지원하기 위해 TDS의 구현체에서 내부 레벨 클래스들이 가져야 할 메소드와 변수를 일반화하여 Level이라는 추상클래스로 작성한다. 그리고 태그 데이타 변환 수행시 자주 자용되는 메소드들을 모아 CoreAPI 클래스로 작성한다(그림 2 참조).

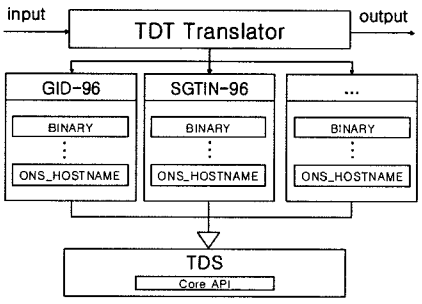

그림 2 변환 수행부 구성도

3.3.1 TDTTranslator

TDTTranslator 클래스는 TDT 표준에서 정의된 변환 인터페이스 translate 메소드를 제공한다. 사용자가 임의의 코드 체계을 입력으로 주더라도 주어진 인자들을 이용하여 자동으로 스킴과 레벨을 검출하고 해당되는 태

그 데이타 변환을 수행할 수 있도록 구성되어 있다.

3.3.2 TDS

TDS는 각 코드체계들에서 공통적으로 사용되는 필드(tagLength, filter 등)와 메소드를 정의하고 있는 추상 클래스이다. 이 클래스의 translate() 메소드는 각 코드체계들의 변환 과정중 일반적인 동작을 정의하고 있는데 변환의 구체적인 동작은 하위 구체클래스에서 구현하도록 추상 메소드로 정의하고 있다. Code Generator에 의해 자동 생성되는 클래스들은 모두 이 TDS를 상속하여 구현되어야 한다.

translate() 메소드는 TDT를 지원하기 위한 가장 핵심적인 부분이며 다음과 같이 코드 스킴에 상관없이 공통적인 변환 절차를 정의하고 있다.

① 입력코드에 대한 전위일치(prefix match) 검색과 정규식(regular expression) 패턴 검사를 통해 입력에 사용된 코드 체계와 레벨을 자동으로 찾아낸다.
② 정규식 패턴을 이용하여 입력 코드로부터 각각의 필드 값을 추출한다.
③ 문자열 조작, 문자변환, 비트열 추가 등의 과정을 통해 입력 필드 값을 출력 필드 값으로 변환한다.
④ 출력 코드에 필요한 부가적인 필드가 있는 경우 그것을 생성한다.
⑤ ABNF 문법을 통해 각각의 필드를 출력 형식에 맞게 조합한다.

translate 메소드의 구현을 위해서는 객체지향 디자인 패턴[6]에서 제시하고 있는 템플릿 메소드(templatge method) 패턴을 적용하여 각 코드 체계별로 차이가 나는 부분은 메소드 오버라이드(override)를 통해 자체적으로 처리하도록 하였다.

3.3.3 Level

각 코드체계의 레벨에 해당되는 Level에 대한 추상클래스를 의미한다. 이 클래스는 레벨별로 이 클래스는 변환과정의 일반적인 동작, 즉 입력된 코드체계의 레벨에 해당하는 옵션을 선택하는 detectOption(), 선택된 옵션으로부터 각각의 필드를 추출하는 extractOption(), 추출된 필드를 이용해 새로운 필드에서 새로운 필드를 추출하는 extract(), 생성된 필드들을 결합하여 새로운 필드를 만들어내는 format(), 출력에 필요한 필드들을 출력 Level의 형식에 맞게 만들어주는 result() 메소드를 가진다.

표 2 Level 클래스의 주요 메소드

메소드	설명
detectOption	각 레벨에 따라 옵션을 찾아낸다.
extractOption	옵션 값을 추출한다.
extract	EXTRACT 룰에 의해 필드값을 추출한다.
format	FORMAT 룰에 의해 필드값을 생성한다.
result	각 옵션의 문법에 맞게 결과를 생성한다.

3.3.4 Option

자바의 Enum타입으로 작성되어 있으며 레벨의 option-Key를 나타낸다. 현 표준에 정의된 바에 따르면 각 레벨은 1개(1)의 옵션을 가지거나 7개(6~12)의 옵션을 가지게 된다.

3.3.5 <SchemeName>_<bit>

자동적으로 생성되는 변환 모듈 클래스로 <코드체계이름_사용비트수.java> 형식으로 Code Generator에 의해 자동적으로 생성된다. 이 변환 클래스들은 추상클래스인 TDS를 상속하며 내부 클래스로 추상클래스 Level을 상속하는 BINARY, TAG_ENCODING, PURE_IDENTITY, LEGACY, ONS_HOSTNAME을 가진다.

3.3.6 TDTTranslationException

TDT 수행 중 예외 상황을 나타내는 클래스들이다. TDT v1.0 표준에 정의된 예외 클래스들을 사용한다.

4. 성능분석 및 평가

제안하는 방식의 실행효율을 기존의 동적수행 방식과 실험을 통해 비교한다. 동적수행 방식은 공개소스 소프트웨어인 Accada의 TDT 모듈을 사용하였다. 이 실험에는 CPU Intel Core 2 Duo 1.8GHz, RAM 1GB 하드웨어와 Microsoft Windows XP 운영체제가 사용되었다. SGTIN-96 코드체계의 4가지 레벨을 입력으로 하여 각각 5가지 레벨로 변환하는 작업을 일정 회수(10,000회) 수행한 뒤, 1회당 평균 수행시간을 계산하였다.

그림 3~그림 6은 실험 결과의 일부를 보인 것이다. 그림에서 Accada와 Proposed는 각각 Accada 오픈 소스와 제안하는 변환기의 실험결과를 나타낸 것이다. 제시된 그림에서 나타난 것처럼 모든 실험 환경에서 제안된 방식을 적용한 것이 Accada 보다 적게는 2배에서 많게는 5배의 속도 차이를 보였다. 또한 Accada의 경우

표 1 TDS 클래스의 주요 메소드

메소드	설명
translate	epcIdentifier, parameterList, outputFormat, Level 을 인자로 입력 받아 변환을 수행하고 결과를 반환한다.
extractParameter	입력된 parameterList를 파싱하여 필드값으로 추출한다.
selectOption	입력된 epcIdentifier로부터 해당 코드체계의 입력 옵션을 선택한다.
findLevel	입력된 epcIdentifier로부터 해당 코드체계의 입력 레벨을 선택한다.
getOutputLevel	입력된 outputFormat으로부터 코드체계의 출력 레벨을 선택한다.

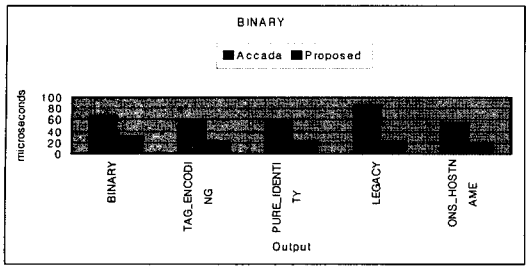

그림 3 BINARY 코드 입력 시 수행 속도 비교

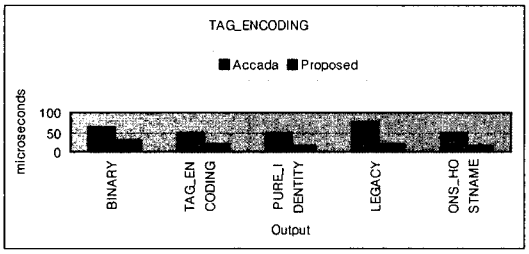

그림 4 TAG_ENCODING 코드 입력 시 수행 속도 비교

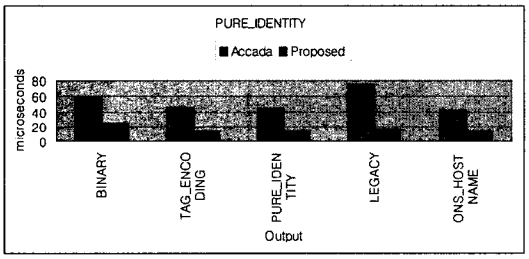

그림 5 PURE_IDENTITY 코드 입력 시 수행 속도 비교

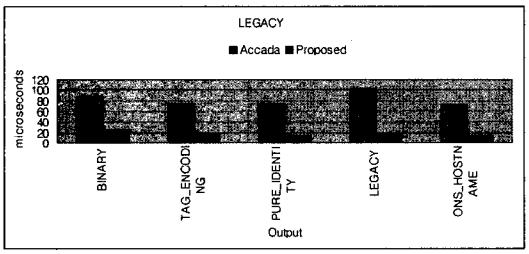

그림 6 LEGACY 코드 입력 시 수행 속도 비교

에는 입력 코드와 출력 코드의 형태에 따라서 수행 속도가 차이를 보이는데 변환과정에서 규칙(rule) 적용이 빈번하게 이루어지는 LEGACY의 경우 두드러지게 나타난다. 반면 제안된 방식에서는 대체로 입력 코드와 출력 코드에 관계없이 비슷한 처리 시간이 소요됨을 보여준다. 이러한 실행 성능의 차이는 앞서 2장 관련연구에서 설명된 것처럼, 동적변환 코드 생성 방식을 사용하는 Accada의 경우에는 실행시에 변수의 바인딩 및 형변환 등을 처리하게 되므로 제안하는 기법에 비해 실행 효율이 떨어지는 점에 기인하는 것으로 볼 수 있다. 특히 LEGACY와 같이 룰의 처리를 위해 문자열 처리함수를 빈번히 사용하는 경우에는, 변수의 검색 및 처리함수의 선택과 호출이 더욱 자주 일어나야 하므로 제안하는 기법에 비해 실행효율이 좋지 않게 된다.

5. 결 론

태그 데이타 변환 기능은 RFID 기반 시스템의 여러 모듈에서 사용되고 빈번하게 수행되므로 수행 효율이 우수해야 한다. 또한 새로운 코드 체계와 변환 코드를 손쉽게 추가할 수도 있어야 한다. 본 논문에서는 TDT v1.0의 명세로부터 태그 데이타 변환 코드를 자동 생성하는 코드 생성기를 제안하였다. 다양한 조합의 태그 데이타 변환 코드를 간단히 개발할 수 있게 하고 변환코드의 유지보수가 편리하다. 기존 오픈 소스 프로그램과의 성능 비교는 코드 생성기에 의해 작성된 변환 코드의 실행 성능이 우수함을 보여준다.

참 고 문 헌

[1] EPCglobal. http://www.epcglobalinc.org
[2] EPCglobal, EPCglobal Generation 1 Tag Data Standards Version 1.1 Rev.1.27 Standard Specification, 2005.
[3] EPCglobal, EPCglobal Tag Data Translation (TDT) 1.0 Ratified Standard Specification, 1-107, 2006.
[4] EPCglobal, The Application Level Events (ALE) Specification, Version 1.0 Ratified Specification, 2005.
[5] Accada. http://accada.org/tdt/index.html
[6] Gamma et. al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley 1995.