XML 데이터의 인라인 바인딩 방법

이 은 정[†] · 유 가 연[††]

요 약

어플리케이션에서 XML 데이터를 이용하기 위한 방법으로 XML 타입 정의에 맞는 클래스를 생성하고 데이터의 인터페이스를 담당하게 하는 XML 바인딩 방법이 있다. 그런데 이러한 방법을 지원하는 기존의 바인딩 프레임워크에서는 XML 정의 문법에서 정의된 모든 요소에 대해 클래스를 생성하여 클래스의 수가 많아지고 전체 어플리케이션의 복잡도가 높아지는 문제가 있다.

본 연구에서는 XML 정의 문법에서 XML 바인딩 클래스 생성이 필요한 요소들을 추출하는 인라인 방법을 제안한다. 제안된 바인딩 클래스 생성 방법은 반복과 재귀 등의 경우에만 클래스를 생성하고 터미널 요소의 값은 필드로 표현하는 클래스를 생성한다. 그리고 인라인된 요소들의 경로를 회복하여 XML 문서를 생성하기 위한 마샬링 알고리즘을 소개한다.

제안된 방법을 검증하기 위하여 IBinder 시스템을 개발하고 생성된 결과를 기존의 방법과 비교하였다. 그 결과 IBinder 시스템에서 생성된 XML 바인딩 클래스의 수가 크게 줄어드는 것을 보일 수 있었다.

키워드 : XML 데이터 바인딩, 인라인 알고리즘, DTD, marshaling

Inline Binding For XML DataInline Binding For XML Data

Eun-Jung Lee[†] · Ga-Yeon Yoo[††]

ABSTRACT

For using XML data in programming languages, there is a data binding method, which generates classes from XML type definitions. However, since existing binding frameworks for this method generate all classes for element definitions, the number of generated classes becomes large and the complexity of the overall application system gets high.

In this research, we propose an inline binding method for selecting necessary classes from element definitions. In the proposed method, classes are created only for elements with repetitions and recursions, and they include fields for values of terminal elements. We introduce a generation algorithm for binding classes and the marshaling methods for recovering the omitted paths.

We develop IBinder system to validate the proposed method and compare the generated codes with the ones of existing systems. As a result, we can show that the number of generated classes decrease substantially compared to other systems.

Key Words : XML data binding, inline algorithm, DTD, marshaling

1. 서 론

XML 데이터가 시스템 간의 데이터 호환과 상호 운용성을 높이기 위한 데이터 교환 수단으로 도입되면서 프로그래밍 언어에서 XML 데이터를 이용하기 위한 접근 방법이 다양하게 제시되었다. 그 중에서 XML 정의에 따르는 XML 바인딩 클래스를 생성하고 이 클래스를 통해 XML 데이터의 인터페이스를 담당하게 하는 XML 데이터 바인딩 방법이 있다[16]. XML 정의에 따라 자동 생성되는 클래스를 *XML 바인딩 클래스*(이하 바인딩 클래스)라 한다. 이 방법은 XML 데이터를 읽어들이고 접근하고 새로운 XML 문서를 생성하는 복잡한 코드를 자동 생성하여 어플리케이션 개발 비용이나 기간의 단축에 상당한 효과를 보이고 있다[13]. 이 방법은 특히 최근 웹서비스의 확산과 함께 XML 메시지 처리 방법으로 대부분의 플랫폼에서 적극적으로 받아들여지고 있다[6, 24].

바인딩 클래스는 데이터를 XML 표현으로 생성(마샬링)하고 XML 데이터에 대응하는 객체를 생성하는(언마샬링) 일을 담당하며, 어플리케이션 코드에서는 XML 데이터에 대응하는 바인딩 클래스의 객체를 통해 바로 XML 데이터를 사용할 수 있다. JAXB나 Castor 등이 대표적인 예인데[16, 17], 바인딩 클래스를 이용하는 방법은 미리 타입이 정해진 XML 문서를 효율적으로 처리할 수 있으며 SAX나 DOM과 같은 낮은 수준의 API에 비하여 데이터의 의미를 그대로 프로그램의 클래스로 반영할 수 있어 사용이 편리하다[19, 20].

† 정 회 원 : 경기대학교 정보과학부 조교수
†† 준 회 원 : 경기대학교 전자계산학과 석사과정
논문접수 : 2005년 10월 24일, 심사완료 : 2005년 12월 19일

기존의 바인딩 프레임워크에서는 XML 정의 문법에서 정의된 구조를 그대로 바인딩 클래스 구조로 반영한다. 우선 XML 문서 정의의 모든 요소들에 대응하는 클래스를 만들 경우 클래스의 수가 무척 커질 수 있다. 또한 XML 문서에서 요소는 데이터의 효율적인 표현보다는 사람이 이해하기 쉬운 데이터 표현에 목적을 두는 경우가 많다. 그러므로 XML의 구조를 그대로 클래스에 반영하는 것이 반드시 바람직한 것은 아니다.

본 논문에서는 바인딩 클래스를 필요한 경우에만 생성하는 방법을 제시하고자 한다. 바인딩 클래스의 중요한 목적은 XML 정의 문법에 나타난 구조를 클래스의 관계(멤버 변수)를 통해 표현하고 데이터의 타입을 변수에 반영하는 것이다. 구조를 표현하는데 필요한 경우에만 바인딩 클래스를 생성하기 위하여 본 논문에서는 관계형 데이터베이스 스키마를 생성하는데 사용되었던 인라이닝 기법을 적용한다[3, 7, 9]. 인라이닝 기법은 DTD로부터 독립적인 테이블을 가져야 할 요소들을 추출하는 방법으로 반복부 및 재귀적인 요소들을 중심으로 테이블을 구성한다. 본 연구에서는 DTD로부터 클래스를 생성할 요소를 추출하기 위한 인라이닝 기법을 제안하고 새로운 바인딩 클래스 생성 방법을 제안한다. 또한 이들 클래스에서 XML 구조를 반영할 수 있는 마샬링 및 언마샬링 방법을 보인다.

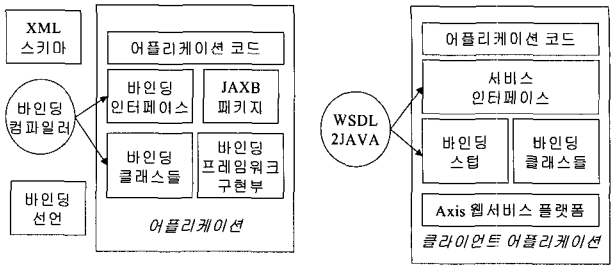
본 논문의 구조는 다음과 같다. 2장에서는 XML 데이터에 대한 다양한 바인딩 방법을 살펴본다. 3장에서는 제안된 새로운 바인딩 클래스 생성 시스템을 설계하고 4장에서 제안된 방법을 구현한 IBinder 시스템에 대한 소개와 함께 타 바인딩 클래스 생성 시스템의 결과와 비교하고 제안된 방법의 장단점을 분석한다. 5장에서 결론을 맺는다.

2. 관련 연구

XML을 이용하는 어플리케이션은 XML 데이터를 입출력하기 위해 강력한 API를 필요로 한다. 프로그래밍 언어에서 XML 데이터를 사용하기 위한 방법으로 가장 먼저 등장한 것은 DOM이나 SAX API다[19, 20]. 이것은 XML 데이터를 트리 자료구조로 표현하는 것으로 간단한 XML 데이터의 접근을 위해 지나치게 많은 프로그램 코드를 필요로 하게 된다.

한편 XML 데이터의 입출력과 접근하는 일을 담당하는 클래스를 이용하는 방법이 있다. 바인딩 클래스는 XML 문서 정의에 따라 자동으로 코드 생성되는데, 이 클래스에서는 XML 데이터를 객체로 바꾸는 일(deserialization 또는 unmarshalling)과 객체를 XML 문서로 변환하는 직렬화(serialization 또는 marshalling)를 지원한다. 또한 어플리케이션의 다른 부분에서 객체의 데이터를 접근하기 위한 인터페이스를 제공하여 어플리케이션에서 별도의 코드 없이 XML 데이터를 접근할 수 있게 지원한다.

바인딩 클래스는 다양한 언어와 환경에서 적용될 수 있는



(가) JAXB의 어플리케이션 구조 (나) Axis 웹서비스 플랫폼의 바인딩 활용 구조

(그림 1) JAXB와 Axis 웹서비스 플랫폼의 바인딩 활용 구조

데, 대표적인 예로 자바 플랫폼에서의 바인딩 프레임워크인 JAXB(Java API for XML Binding)가 있다[16]. 바인딩 프레임워크는 바인딩을 위한 클래스 코드를 스키마나 DTD와 같은 XML 정의로부터 자동으로 생성한다. 생성된 바인딩 클래스를 이용하면 어플리케이션에서는 빠른 프로토타입 개발이 가능하다.

한편 웹서비스 환경에서는 XML 기반의 SOAP 메시지를 생성하여 송신하고 수신하여 처리하는 과정을 바인딩 클래스가 담당해 주고 있다[6, 24]. 여기서는 기존의 통신 스텁 클래스가 하던 직렬화의 역할을 확장하여 바인딩 클래스가 데이터를 XML 형태로 직렬화하고 SOAP 플랫폼이 이진 데이터 스트림으로 직렬화하는 일을 나누어 담당한다.

클래스를 생성하지 않고 어플리케이션에 속하는 원래의 클래스에 대해 바인딩을 위한 매핑을 지정하는 방법도 있다. Castor가 대표적인 예인데[17], 여기서는 XML 데이터와 클래스와의 데이터 변환을 위한 코드가 자동 생성되어 클래스 자체는 어플리케이션에 포함된 것을 사용할 수 있다. 이외에도 DOM 트리를 이용하면서 데이터 접근 메소드를 지원하는 façade 클래스 형태의 JDOM이나[21] SAX 형태의 이벤트 기반 XML 처리 인터페이스를 XML 스키마 정의에 따라 생성하는 방법도 제안되었다[11]. 또한 웹서비스의 지원을 위한 프로그래밍 언어의 바인딩 클래스 생성에 관한 연구도 있다[6, 10]. 정적 바인딩 방법과 달리 XML 지원 타입을 내재한 새로운 프로그래밍 언어가 제안되기도 했고[7, 8], 함수 또는 논리 언어에서의 XML 데이터 바인딩 방법도 연구되었다[4, 5].

프로그래밍 언어에서 XML을 지원하기 위한 다양한 접근 방법 중에서 본 연구에서는 정적인 바인딩 클래스의 생성 방법을 다룬다. 바인딩 클래스가 XML 데이터 타입을 코드화하여 XML이 가지는 유연성이나 자유로운 데이터 형식의 장점이 없어질 뿐 아니라 생성된 코드를 이용하는 어플리케이션 코드의 모듈화나 재사용성이 떨어지게 된다는 비판도 있다[7, 8]. 그러나 이러한 문제점들에도 불구하고 미리 정해진 XML 문서 타입으로 동작하는 시스템에서 정적 바인딩 방식은 효율적이고 사용이 편리하여 많이 사용되고 있으며 웹서비스 개발 환경에서는 일반화된 프레임워크로 받아들여지고 있다.

3. 인라인에 의한 바인딩 클래스 생성

3.1 인라인의 도입 방법

바인딩(binding)에는 여러 가지 의미가 있으나, 여기서의 바인딩이란 바인딩 클래스를 이용하여 XML 데이터를 프로그램의 객체와 연결하는 과정으로 한다. 즉 XML 데이터를 프로그램에서 사용할 수 있도록 객체로 변형하는 과정이라 할 수 있다. 그래서 바인딩 클래스는 XML 정의 문법에서 정의된 구조를 프로그램에서 객체의 데이터와 객체 간의 관계로 표현할 수 있어야 한다.

기존의 바인딩 프레임워크에서는 XML 정의 문법의 모든 요소에 대해 대응하는 바인딩 클래스를 생성하였다. 바인딩 클래스의 개수와 구조의 복잡도는 통합을 위한 부담과 함께 전체 어플리케이션의 복잡도에도 영향을 미치게 된다. 그러므로 더 간단하고 적은 수의 바인딩 클래스로 트리의 구조를 표현할 수 있다면 바람직할 것이다. 본 논문에서는 XML 데이터의 구조를 반영하기 위하여 꼭 필요한 바인딩 클래스만 생성하는 방법을 제시하고자 한다.

다음 예를 통해 바인딩 클래스의 생성이 필요하지 않은 요소의 경우를 살펴보자. XML 데이터의 구조를 프로그램 객체에 반영하기 위하여 JAXB 등 기존의 바인딩 프레임워크에서는 요소의 부모 자식 관계를 그대로 객체의 관계로 표현한다. 다음 예에서 <주소> 요소에 대응하는 클래스는 자식 요소인 <시>, <도>, <세부주소> 요소에 대응하는 필드를 가지고 <세부주소>는 다른 클래스에 대응된다. 여기서 생성규칙이 나타나 있지 않은 요소는 단순 터미널 노드이다. <세부주소> 요소는 값이나 속성을 가지지 않고 단순히 태그를 생성하는 역할만 한다. 물론 XML 정의 문법을 다시 작성하여 <세부주소> 요소를 없애고 그 자식들을 바로 <주소> 요소의 자식으로 나타나게 하는 것은 가능하다. 그러나 본 논문에서는 XML 정의 문법을 바꾸지 않고 주어진 문법에서 필요한 클래스와 클래스 간의 관계 만을 표현하는 방법을 찾고자 한다.

```
회원명부 -> 회원*
회원 -> 이름 주소 학력 소속
소속 -> 기관명 직위 주소
주소 -> 도 시 우편번호 세부주소
세부주소 -> 동 번지 상세주소
학력 -> 학위*
```

XML 정의 문법에서 클래스를 생성하여야 할 요소들을 추출하는 방법으로 DTD에서 데이터베이스의 관계형 테이블을 추출하기 위하여 사용하였던 인라이닝 방법을 도입하고자 한다[3, 9, 12]. 인라이닝은 계층적 구조의 XML 정의 문법에서 중복을 최소화하면서 필요한 관계형 테이블을 생성하는 알고리즘이다. 인라이닝은 단순 인라이닝과 공유 인라이닝 알고리즘이 있다. 단순 인라이닝은 반복부에 대해서만 별도의 테이블을 만드는 방법이고 공유 인라이닝은 반복부와 한번 이상 사용되는 요소에 대해 별도의 테이블을 만드는 방법이다. 위의 예에서는 단순 인라이닝 방법에서는 <회원>과 <학력>이 테이블이 되고 공유 인라이닝으로는 이외에 추가로 <주소> 테이블이 생성된다. Lu 등은 공유 인라이닝에 추가적인 최적화 방법을 소개하였다[9].

본 연구에서는 위의 인라이닝 방법을 도입하여 반복이나 공유 등 필요한 경우에만 별도의 클래스를 생성하는 방법을 소개한다. 이것은 대부분의 바인딩 프레임워크에서 사용하는 디폴트 규칙과 상당히 큰 차이가 있는데, 앞에서 살펴보았듯이 바인딩 프레임워크에서는 단순 값이 아닌 모든 요소는 클래스로 생성한다.

필요한 경우에만 클래스를 생성하기 위해서는 몇 가지 문제를 해결해야 한다. 우선 클래스 생성이 필요한 경우가 언제인가를 정의할 수 있어야 한다. 다음으로 생성된 클래스에서 어떻게 XML 트리의 전체 구조를 표현할 것인가를 설계하여야 하며, 또한 마샬링/언마샬링에서 XML 문서의 입력 및 생성을 위한 알고리즘을 제시하여야 한다. 아래에서는 이상의 단계에 대하여 구체적으로 살펴본다.

3.2 인라이닝 클래스의 선정

이 절에서는 DTD로 주어진 XML 정의 문법으로부터 인라이닝에 의해 클래스를 생성할 요소를 결정하는 방법을 살펴본다. 클래스로 생성할 요소를 구분하기 위해 DTD (방향) 그래프를 사용한다. DTD 방향 그래프는 다음과 같이 정의된다.

[정의 1] DTD 그래프 G의 노드는 DTD에서 정의된 요소와 속성들의 집합이고 노드들 간의 에지는 다음과 같이 정의된다.

(i) DTD의 생성 규칙에서 좌변에 있는 요소에서 우변의 요소들로 에지를 가진다.
(ii) 속성은 해당하는 요소로부터 에지를 가진다.
(iii) 우변이 #PCDATA인 경우는 에지를 만들지 않는다.

<예 1>의 DTD에 대해서 DTD 그래프가 (그림 2)의 (가)에 나와있다.

$R \rightarrow HA$	$C \rightarrow C_1(C_2C_3)*$
$A \rightarrow B(CD)*$	$D \rightarrow D_1D_2$
$B \rightarrow B_1B_2$	$D_2 \rightarrow EF*$

〈예 1〉 논문에서 사용할 예제 DTD 코드

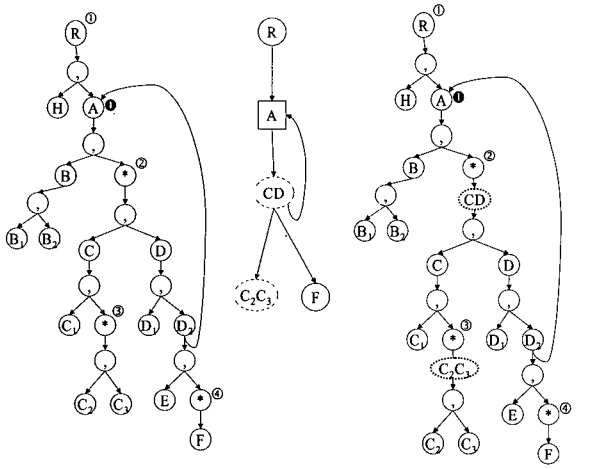
터미널 노드만 값을 가지도록 하기 위하여 우변이 #PCDATA인 요소에 속성이 있는 경우는 해당 값을 가지는 터미널 요소 노드를 추가하였다. DTD에서 자식 요소들 간의 순서가 있으므로 DTD 그래프에서 나가는 에지들에 순서를 정할 수 있다. DTD 그래프에서 노드의 종류는 다음과 같이 나누어 볼 수 있다.

(i) 내부 노드 : 나가는 에지를 가짐
  1. 루트 요소 노드
  2. 내부 기호 노드 : (*), (+), (?), (,), (|) 노드
  3. 내부 요소 노드
(ii) 터미널 노드 : 나가는 에지가 없는 노드로 값의 타입을 가짐
  1. 터미널 요소 노드
  2. 속성 노드

내부 요소 노드 중에서 자기 자신으로 돌아오는 경로가 있는 경우를 *재귀* 노드라고 한다. 또한 내부 기호 노드 중에서 (*), (+), (?) 노드를 *반복* 노드라고 한다. (그림 2)에서 반복 노드는 ②, ③, ④번의 세 개이고, 루트노드에 대한 가상의 반복 노드를 ①번으로 표시했다. 한편 재귀 노드는 ❶번이다.

DTD에서 모든 반복 노드(생략 노드 포함)가 하나의 요소 노드만 자식으로 가지면 그 DTD가 반복부 팩토링되었다고 한다[2]. 주어진 DTD를 반복부 팩토링하기 위하여 반복 노드에 대해 자식 노드가 요소 노드가 아닌 경우 가상 자식 요소 노드를 추가한다. (그림2)의 예에서 ①과 ④의 노드는 자식이 요소 노드이고 ②, ③은 자식이 기호 노드이다. 그러므로 반복부 팩토링을 위해 가상 자식 요소 노드를 추가할 수 있는데, (그림 2)의 (나)에서 점선 타원으로 표시된 CD 노드와 C_2C_3 노드가 이들이다.

본 연구에서는 Shanmungasundram 등의 인라이닝 방법을 적용하기 위하여 몇 가지 사항을 고려하였다. 클래스 생성에서 공유는 동일한 필드를 다른 클래스에서 포함하는 것으로 데이터 중복의 문제가 발생하지 않으므로 본 연구에서는 단순 인라이닝 방법을 기본으로 한다. 한편 원래의 방법에서는 생략(DTD 그래프에서 (?) 이하의 부분)을 (*) 반복으로 단순화하여 처리하였으나 본 연구에서는 터미널 노드가 아닌 생략부는 별도로 고려하여 별도의 클래스 생성 대상으로 본다.



(가) 원래의 DTD 그래프 (나) 반복부 팩토링된 DTD 그래프

(그림 2) <예 1>에 대한 DTD 그래프

[정의 2] 주어진 반복부 팩토링된 DTD 그래프 G에 대해 클래스를 생성하여야 하는 노드들의 집합 $V_{class}(G)$는 다음과 같은 원소들을 포함한다.

(1) 루트노드
(2) 내부 노드 중 반복 또는 생략 노드의 자식 요소 노드 (반복부 노드)
(3) 재귀 노드
(4) ID를 속성으로 가지는 노드

이상의 4가지 종류의 노드들은 서로 교집합을 가질 수 있는데, $V_{class}(G)$는 이들 노드를 모두 합한 것으로 DTD 그래프 G가 분명할 때는 V_{class}로 표시한다. 예를 들어 (그림 2)에서 V_{class} = {R, A, CD, C_2C_3, F}이다. V_{class}는 클래스를 생성해야 할 요소를 나타내는데, 이중에서 ID와 IDREF에 의해 표현되는 노드들의 처리는 JAXB 등 기존의 바인딩 프레임워크에서 사용하는 방법과 동일하므로 이하에서는 고려하지 않는다.

4. 인라이닝에 의한 바인딩 클래스 생성 방법

4.1 반복부 그래프

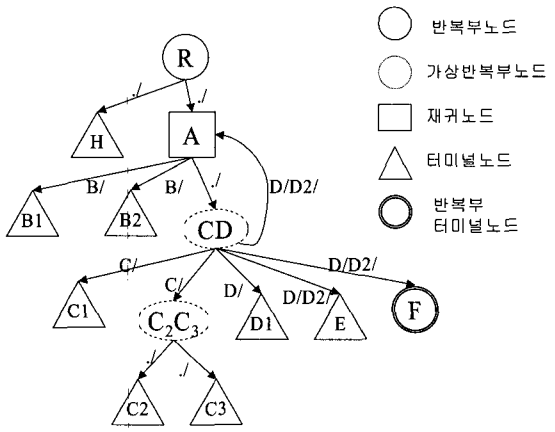
DTD 그래프에서 V_{class}에 속하는 노드들이 클래스 생성 대상이므로 이 노드들을 중심으로 반복부 그래프를 구성한다. 터미널 노드의 값이 프로그램 객체에 바인딩할 수 있어야 하므로 값을 가지는 터미널 노드에 대응하는 필드가 바인딩 클래스에 포함된다. 한편 V_{class}에 속하지 않는 내부 노드들은 태그를 생성하여 XML 트리 상의 경로를 표현하는 역할만 하게 되므로 바인딩 클래스에는 표현되지 않는다.

[정의 3] 주어진 *DTD* 그래프 G와 V_{class} 집합에 대해 반복구조 그래프 $G^* = (V^*, E^*)$는 다음과 같이 정의된다.

$V^* = V_{class} \cup V^*_T$, V^*_T = *G의 터미널 노드 집합*
$E^* = \{(v, w) \mid v, w \in V_{class}$, $path_G(v, w)$*가* V_{class}*에 속하는 노드를 포함하지 않음*$\}$
$\cup \{(v, x) \mid v \in V(G), x \in V^*_T$, $path_G(v,w)$*가* V_{class}*에 속하는 노드를 포함하지 않음*$\}$

$path_G(v, w)$는 DTD 그래프 G에서 두 노드 간의 경로이고 레이블의 연속으로 표시된다. G^*에서 터미널 노드의 집합을 V^*_T로 표현하고 V^*_T에 속하는 노드 중 반복부 노드인 것을 반복부 터미널 노드라고 부른다. 한편 내부 노드들은 내부 반복부 노드, 가상 반복부 노드, 재귀 또는 생략 노드로 분류할 수 있다. 노드 v의 노드 종류를 $node_type(v)$로 표시한다.

G^*에서 내부 노드 v에 대해 나가는 에지로 연결된 다음 노드들은 DTD 그래프에서의 순서를 인덱스로 가진다. $succ_i(v)$는 노드 v에 대해 다음 노드들 중 인덱스가 i인 것을 표시한다. 다음 노드의 순서 정보는 객체의 표현에서는



(그림 3) <예 1>에 대한 반복부 그래프

중요하지 않으나 마샬링할 때 XML 트리의 자식들이 순서를 가져야 하므로 필요하다. 이외에 터미널 노드 v의 값이 가지는 타입은 value_type(v)로 표시한다. 축약된 중간 노드들의 태그를 각 에지에 대응하는 경로로 기억하는데, G^*에서 $(v, w) \in E^*$에 대해 원래 DTD 그래프에서의 두 노드 간의 상대 경로를 $relative_path(v, w)$로 표현한다.

(그림 3)은 <예1>에 대응하는 G^* 그래프를 보여준다. 여기서 R은 내부 반복부 노드이고 CD와 C_2C_3는 가상 반복부 노드, F는 반복부 터미널 노드이고 A는 재귀 노드이다. 반복부 그래프는 클래스의 관계와 각 클래스가 나타낼 필드의 값을 표현하므로 이 그래프로부터 클래스 코드가 바로 생성될 수 있다.

4.2 클래스 코드 생성 알고리즘

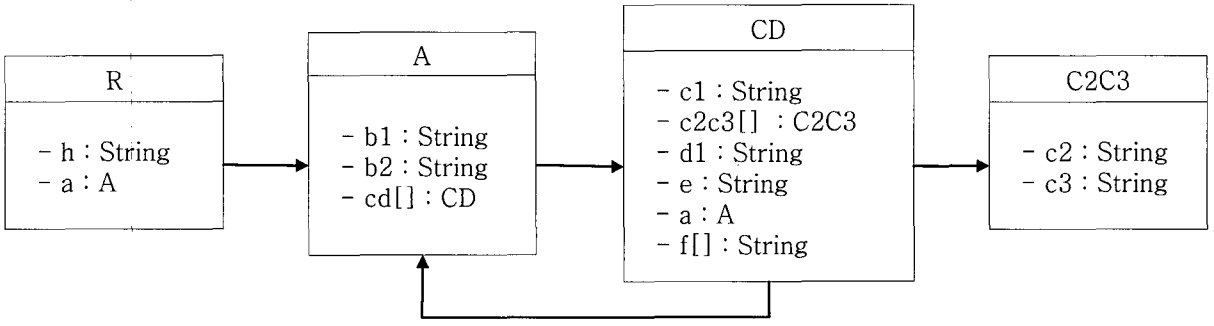
반복부 그래프의 모든 내부 노드에 대해 대응하는 클래스가 생성된다. 이들 클래스에는 노드의 나가는 에지에 대응하는 필드가 포함되는데, 다음 노드가 터미널 노드이면 단순 자료형의 필드가 생성되고 내부 노드이면 해당 클래스의 객체 필드가 생성된다.

다음 알고리즘은 반복부 그래프에서 클래스 코드를 생성하는 과정을 보여준다. 이 알고리즘은 반복부 그래프 G*의 각 노드에 대해 재귀적으로 순회하면서 호출되는데, 노드 v에 대해 그 노드에 대응하는 필드가 생성되며, 새로 생성된 필드가 포함될 클래스는 호출 전에 미리 생성되어 매개변수로 전달된다. 코드 생성 알고리즘은 문서 전체를 포함할 클래스 C를 먼저 생성하여 반복부 그래프의 루트 노드 root와 함께 *Generate_class_codes(root, C)*로 호출한다.

코드 생성 알고리즘에서 (1), (2)의 단계는 클래스를 생성하지 않고 단순값 타입의 필드를 생성하는 경우이다. 그 중에서 현재 노드가 반복부 터미널 노드인 경우는 배열이 생성되어야 한다. (3) 단계는 클래스가 생성되는 내부 노드에 대응하는데 (3)-(A) 단계는 클래스가 이미 생성된 재귀 노드에 대해 코드 생성을 반복하지 않고 종료한다. (3)-(B) 단계는 대응하는 클래스를 생성하고 재귀적으로 다음 노드를 방문하면서 필드를 추가한다.

```
[알고리즘] generate_class_codes(v,W)
반복부 그래프 G*
v : 이 함수를 호출할 때의 현재 노드
W : v의 이전 노드 w에 대응하는 클래스를 W
    (1) v가 단순 터미널 노드이면
        (A) value_type(v)의 필드를 생성하여 클래스 W에
            필드로 추가한다.
    (2) v가 반복부 터미널 노드이면
        (A) value_type(v)의 배열을 선언하고 클래스 W에
            필드로 추가한다.
    (3) v ∈ V_class
        (A) 노드 v에 대응하는 클래스가 이미 있으면
            (재귀 노드인 경우)
            i. 이 클래스 객체를 클래스 W에 필드로
               추가한다.
        (B) 아직 대응하는 클래스가 없는 경우
            i. label(v)로 클래스를 생성하여 v에 대응하는
               클래스 V로 등록한다.
            ii. v가 반복부 자식 요소이면
                1. V 타입의 객체 배열을 클래스 W에
                   필드로 추가한다.
            iii. v가 생략 또는 재귀 요소이면
                1. V 타입의 객체를 클래스 W에 필드로
                   추가한다.
            iv. v의 자식들 v1, v2, .., vn을 순서대로
                1. generate_class_code(vi, V)로 호출한다.
```

(그림 4)는 <예 1>의 DTD에 대해 생성되는 클래스 구조를 보여준다. 클래스 R에서 다음 노드 H에 대응하는 변수 h를 추가하고 클래스 A 타입의 객체 a의 필드를 추가한다. 클래스 A를 생성한 후 노드 A의 다음 노드들을 방문하면서 클래스 A의 필드를 추가한다.



(그림 4) <예 1>의 DTD에 대해 생성된 바인딩 클래스 구조

생성된 클래스의 각 필드에 대해서는 대응하는 접근 메소드가 정의되는데 본 논문에서는 단순 필드인 경우는 get/set, 배열의 경우는 add/remove, 생략에 대응하는 필드인 경우는 delete/insert를 추가하였다. 접근 메소드의 생성은 기존의 방법과 차이가 없으므로 생략한다.

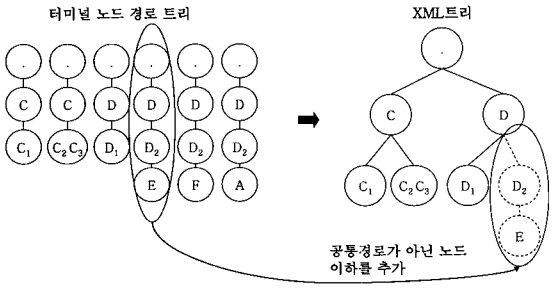
4.3 마샬링 및 언마샬링 메소드 코드 생성

기존의 바인딩 프레임워크에서 마샬링 및 언마샬링은 트리 구조에 클래스 구조가 직접 대응되므로 터미널 노드에 대한 해당 필드 값을 바인딩하는 것이 주를 이루었다. 그러나 본 논문에서 제안한 바인딩 클래스 관계는 트리의 구조를 그대로 따르지 않으므로 클래스를 생성하지 않은 내부 요소 노드의 처리가 고려되어야 한다.

언마샬 단계는 정해진 순서에 따라 트리를 순회하면서 필요한 데이터를 해당 필드에 저장하고 마샬 단계는 인라이닝에 의해 변경된 부분을 회복하면서 XML 문서를 생성한다. 아래에서는 반복부 그래프에 저장된 순서 및 경로 정보로부터 대응하는 XML 문서를 생성하는 마샬링 함수의 코드 생성 알고리즘을 살펴본다. 터미널 노드 데이터의 자료형에 따른 마샬링 방법은 기존의 바인딩 프레임워크와 동일하므로 여기서는 고려하지 않는다.

클래스를 생성하는 내부 노드나 필드를 생성한 터미널 노드들은 각각 DTD 그래프에서 축약된 부분을 상대 경로로 가지고 있다. Relative_path(v, w)가 축약된 경로인데 마샬링에서는 결과 트리에 이 경로를 추가하여 노드를 생성한다. 상대 경로를 생성 중인 XML 트리에 추가할 때 공통된 최대 경로 만큼 내려가서 추가하는 것을 *덧붙인다*고 한다. (그림 5)는 ./D/D2/E를 덧붙이기 위해 최대 공통 경로인 ./D 아래에 D2 노드를 생성하여 추가하는 것을 보여준다.

마샬링 메소드는 축약된 경로를 회복하면서 XML 결과 트리를 생성하는데, 해당 클래스의 필드들에 대응하는 XML 요소와 값을 트리로 생성한다. 또한 객체 필드에 대해 재귀적으로 마샬링 메소드를 호출한다. 이상과 같은 마샬링 코드는 반복부 그래프를 참조하여 생성될 수 있다.



(그림 5) 상대경로를 생성하는 트리에 덧붙이기

[알고리즘] *클래스A.marshal(현재노드x)*
이 객체의 마샬링을 호출한 상태에서 생성된 트리 T_{new}의 현재 노드가 x라 하자.
클래스 A에 대응하는 반복구조 그래프의 노드를 v라 하자.
1. 나가는 에지의 index 값에 따라 차례대로 다음을 수행한다.
   (A) $w = succ_i(v)$라 하자.
      i. *relative_path(v,w)*를 x의 자식 트리에 덧붙인다. 덧붙인 제일 마지막 노드를 y라 하자.
   (B) w가 단순 터미널 노드이면
      i. 요소 w를 생성하고 y의 자식으로 추가한다.
      ii. 필드의 값을 마샬링하여 추가한 노드의 값으로 추가한다.
   (C) w가 반복부 터미널 노드이면
      i. w에 대응하는 배열을 ws라 하면 배열의 각 객체 ws[i]에 대해
         1. 요소 w를 생성하고 y의 자식으로 추가한다.
         2. 필드의 값을 마샬링하여 생성된 요소의 값으로 추가한다.
   (D) w가 내부 반복부 노드이면
      i. w에 대응하는 객체 배열을 ws라 하면 배열의 각 객체 ws[i]에 대해
         1. w가 가상노드가 아니면
            A. w에 대응하는 자식 요소를 생성하여 y의 자식으로 추가한다. 추가한 노드를 y라 한다.
         2. y를 현재 노드로 마샬링 함수 ws[i]->marshall(y)를 호출한다.
   (E) w가 재귀 또는 생략 클래스 노드이면
      i. 해당 필드 객체에 대해 y를 현재 노드로 마샬링 함수를 호출한다.

```
Public void marshal() {
   marshalTermNode("C1", c1);
   attachPath("C2C3");
   for (int i=0; i < c2c3.size(); i++)
      ((C2C3)c2c3.get(i)).marshal();
   marshalTermNode("D1", d1);
   marshalTermNode("E", e);
   marshalRepeatTermNode("F", f);
   if (a != null){
      attachPath("A");
       a.marshal();
}
```

(가) 클래스 CD의 marshal 메소드

```
static Node head, cur_X;
public void attachPath(String name) {
   head = createPath(getPath(name));
   cur_X = attachPath(cur_X, head); // 경로덧붙이기
}
public void
marshalTermNode(String name, String value) {
   attachPath(name);
   cur_X.addChild(new Node(name, TERM, value));
}
public void
marshalRepeatTermNode(String name, Vector valueS)
{
   attachPath(name);
   for (int k=0;k<valueS.size();k++){
      cur_X.addChild(new
            Node(name, TERM, (String)valueS.get(k)));
   }
```

(나) IBinder 클래스의 메소드

(그림 6) <예 1>의 클래스 CD에 대한 생성 코드 및 베이스 클래스 관련 메소드

실제 생성된 코드를 살펴보기 위해 <예 1>의 DTD에 대해 가상 요소 <CD>에 대응하는 클래스의 마샬 함수를 보면 (그림 6)과 같다. 모든 바인딩 클래스는 베이스 클래스 IBinder를 상속한다. 반복부 그래프에서 <CD>에 대응하는 노드의 나가는 에지 순서는 C1, C2C3*, D1, E, F*, A이다. 클래스 CD의 필드들은 반복부 그래프의 에지에 대응하는데 각 에지의 다음 노드 종류가 단순 터미널 노드 및 반복부 터미널 노드인 경우에는 IBinder 클래스의 메소드인 marshalTermNode() 또는 marshalRepeatTermNode()를 호출한다.

5. 결과 비교 및 분석

본 논문에서 제안된 인라인 바인딩 방법은 생성되는 바인딩 클래스 수를 현저하게 줄일 수 있으며 불필요한 클래스 구조를 생략하여 어플리케이션에서 좀더 간단하게 바인딩 클래스를 통합할 수 있다. <표 1>은 제안된 방법을 다른 바인딩 프레임워크에서 생성된 클래스 수와 비교한 결과이다.

JAXB[16]와 Axis 플랫폼의 WSDL2Java 시스템[24]은 XML 정의 문법으로 스키마를 이용하므로 DTD를 XML 스파이[23]를 이용하여 스키마로 자동 변환시켜 바인딩 클래스를 생성하였다. 스키마로 변환하면서 추가적인 클래스 생성 요인이 발생하지 않도록 불필요한 요소 타입 정의는 내부 요소 타입으로 수정하였다. JAXB는 인터페이스를 생성하고 WSDL2Java과 본 논문의 IBinder 시스템은 클래스를 생성하는데, 여기서는 구분하지 않고 생성된 인터페이스 또는 클래스의 개수로 비교하였다.

JAXB와 WSDL2Java 시스템은 유사한 클래스 생성 방법을 사용하는데 아래 표에서 보여지듯이 JAXB는 WSDL2Java 시스템에 비하여 클래스의 개수가 더 많다. 그 이유는 공유된 터미널 요소 정의에 대해서 JAXB에서는 클래스를 생성하는 반면 WSDL2Java 시스템은 클래스를 생성하지 않는다. 또한 반복부의 처리에서도 차이가 나는데, JAXB는 반복부로 묶여있는 요소에 대해서는 기본 타입 인터페이스와 요소 인터페이스를 중복하여 생성한다. 그 결과 JAXB의 생성 클래스 수가 WSDL2Java 시스템에 비하여 더 많다.

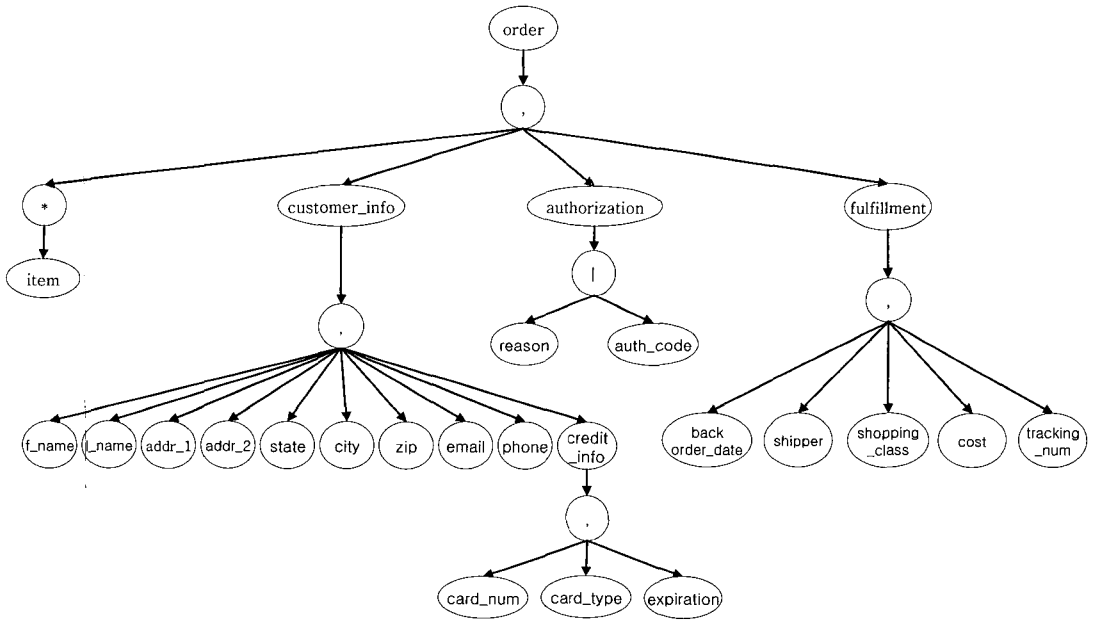
〈표 1〉

문서명	요 소 개 수	JAXB 생성 클래스 수	Axis 웹서비스 플랫폼	IBinder 생성 클래스 수
Order.dtd	26	5	5	1
project.dtd	18	10	5	4
transcript.dtd	49	17	14	6
Submission.dtd	60	25	13	3
packet.dtd	44	14	11	7

[출처] 1) http://www.leaderit.ru/books/e-Shop/Cod/c05/order.dtd
2) http://www.cs.bham.ac.uk/~aps/teaching/projects/XML/projects.dtd
3) http://www.cs.uwm.edu/classes/cs790/digdoc-s2003/assignments/common-transcript.dtd
4) http://oda.state.or.us/purs/eds/purs.dtd
5) http://www.panopticsearch.com/xml/padre_results_v5.5.dtd

IBinder 시스템의 클래스 생성 결과를 타 시스템과 비교하기 위하여 위의 비교 대상 DTD 중에서 order.dtd를 살펴본다. DTD 그래프가 (그림 6)에 나와있는데, 여기서 IBinder 시스템은 order에 대응하는 클래스만 생성하고, 타 시스템에서는 order, customer_info, authorization, credit_info와 fulfillment 등 5개의 클래스가 생성된다.

제안된 방법으로 생성된 바인딩 클래스는 임의의 XML 문서를 읽어들이고 수정된 결과 XML 문서를 생성하는 언마샬링/마샬링 기능을 지원한다. 그러므로 어플리케이션에서는 적은 수의 클래스 만으로도 XML 데이터를 객체에 바인딩하여 사용할 수 있다. 그러나 JAXB 등 기존의 방법이



(그림 6) order.dtd의 DTD 그래프

XML 문서의 구조나 요소를 그대로 클래스에 반영하는데 비해 제안된 방법에서는 인라인된 요소는 클래스 구조에 표현되지 않으므로 XML 문서 상의 경로가 클래스 코드에서 바로 적용되기는 어렵다. 그러므로 사용자가 XML 데이터를 접근하기가 불편할 수 있는데, 예를 들면 XPath를 이용하는 데이터 접근이나 검색 질의를 처리하는데 추가적으로 노력이 필요할 것이다. 본 연구에서는 생성된 인라인된 바인딩 클래스의 사용에 관한 조사와 함께 개선방안을 제안할 계획이다.

6. 결 론

본 논문에서는 DTD로부터 XML 데이터 바인딩 클래스를 생성하기 위하여 필요한 경우에만 클래스를 생성하는 인라인 기반의 바인딩 방법을 제안하였다. 제안된 방법은 XML 정의 문법에서 관계형 데이터베이스 테이블을 추출하기 위한 인라인 방법을 적용하여 반복과 재귀 및 생략의 경우에 대해서만 클래스를 생성하고 터미널 노드 값을 필드로 포함한다. 제안된 인라인 바인딩 방법에 기반하여 바인딩 클래스 코드 생성 알고리즘을 제안하였고 마샬링 과정에서 생략된 경로를 회복하여 생성되는 트리에 표현하는 방법을 소개하였다.

제안된 방법의 타당성을 보이기 위하여 IBinder 시스템을 개발하고 생성된 결과 코드를 타 바인딩 플랫폼의 결과와 비교하였다. 그 결과 제안된 방법은 다른 시스템에 비해 현저하게 적은 수의 클래스를 생성함을 보였다.

본 논문에서 제안된 인라인 기반 바인딩 방법은 기존의 방법과 결합하여 사용자가 원하는 경우 최소의 클래스만을 생성하는 것을 가능하게 해 줄 것이다. 현재 개발된 시스템에서 생성된 코드의 클래스 구조가 원래 XML 정의 문법의 구조와 달라져 개발자가 사용하기 어려운 점이 있는데, 향후 연구에서는 이러한 점을 개선하고 기존의 방법과 결합하여 개선된 바인딩 시스템을 개발할 계획이다. 또한 XML 스키마를 지원하도록 IBinder 시스템을 확장할 계획이다.

참 고 문 헌

[1] 김경덕, "온라인 대화 행위에서 XML 기반 메시지를 이용한 미디어 지원", 한국정보처리학회 논문지 11(B)권 1호, pp.91-98, 2004.

[2] 이은정, "XML 데이터 공유를 위한 리스트 잠금 프로토콜", 한국정보처리학회 논문지 11(D)권 7호, pp.1367-1374, 2004.

[3] 조정길, "인라이닝에 기반한 XML 스키마의 관계형 스키마 변환 기법", 한국정보처리학회 논문지 11(D)권 5호, pp.1021-1030, 2004.

[4] Frank Atanassow, et.al.,. "Scripting XML with Generic Haskell", Technical Report UU-CS-2003.

[5] Jorge Coelho and Mario Florido. "Type-based XML processing in logic programming," In PADL 2003, pp.273-285, 2003.

[6] R. Engelen, et.al, "Developing web services for C and C++," IEEE Internet Computing 7(2), pp.53-61, 2003.

[7] M. Harren, et.al, "XJ: integration of XML processing into Java," In Proceedings of the 13th International World Wide Web conference on Alternate track papers & posters, pp.340-341, 2004.

[8] H. Hosoya, et.al, "Xduce: A statically typed XML processing language," ACM TOIT 3(2), 2003.

[9] S. Lu, et.al, "A New Inlining Algorithm for Mapping XML DTDs to Relational Schemas," ER'2003, 2003.

[10] T. Milo, S. et.al, "Exchanging Intensional XML Data," In Proc. of ACM SIGMOD, San Diego, 2003

[11] F.Reuter, N.Luttenberger, "STAX/bc: A binding compiler for event-based XML data binding APIs," PLANX-2004, pp. 64-72, 2004.

[12] J..Shanmugasundaram, et.al, "Relational databases for querying XML documents: Limitations and opportunities," VLDB Journal, pp.302-314, 1999.

[13] F.Simeoni, et.al, "Language bindings to XML," IEEE Internet Computing 7(1), pp.19-27, 2003.

[14] XML Data Binding Resources. http://www.rpbourret.com/xml/XMLDataBinding.htm.

[15] XML and Java technologies: Data binding, http://www-106.ibm.com/developerworks/xml/library/x-databdop/.

[16] Java Architecture for XML Binding (JAXB) http://developer.java.sun.com/developer/technicalArticles/WebServices/jaxb/.

[17] Castor. http://www.castor.org.

[18] jbind. http://jbind.sourceforge.net/.

[19] SAX. http://www.saxproject.org.

[20] DOM. http://www.w3.org/DOM.

[21] JDOM. http://www.jdom.org/.

[22] XML Schema. http://www.w3.org/XML/Schema.

[23] XML Spy, http://www.xmlspy.com.

[24] Axis WSDL2Java, //ws.apache.org/axis.

이 은 정

e-mail : ejlee@kyonggi.ac.kr

1988년 서울대학교 계산통계학과(학사)

1990년 한국과학기술원 전산학과(공학석사)

1994년 한국과학기술원 전산학과(공학박사)

1994년~2000년 한국전자통신연구원 선임연구원

2001년~현재 경기대학교 정보과학부 조교수

관심분야 : 컴파일러 이론, XML 처리 기술 등

유 가 연

e-mail : despoin@kyonggi.ac.kr

2006년 경기대학교 정보과학부(학사)

2006년~현재 경기대학교 전자계산학과 석사과정

관심분야 : XML, 웹 서비스