XML DTD 모호성 제거와 DOM 활용을 위한 디자인 패턴 연구

김태현○ 고승규 최윤철

연세대학교 컴퓨터과학과
{superkth, pitta, ycchoy}@rainbow.yonsei.ac.kr

A Study of XML Design Patterns for Removing Ambiguity in DTD and Using DOM in XML Application

Tae-Hyun Kim Seung-Kyu Ko Yoon-Chul Choy
Dept. of Computer Science, Yonsei University

요 약

XML(eXtensible Markup Language) 은 차세대 인터넷 기술의 대표적인 기술 요소로서 현재 놀라운 속도로 발전하고 있으며 다양한 인터넷 응용분야에서 적용하고 있다. 이런 추세는 XML 어플리케이션을 복잡화, 대형화 시키고 있으며 이로 인하여 XML 어프리케이션 개발과 유지 보수가 점점 어려워지고 있다. 이러한 문제점을 해결키 위해 객체지향 개발 방법론에 적용되어온 디자인 패턴(Design Patterns) 개념을 XML 개발 관련 분야에 적용시키는 연구가 이루어지고 있다. 본 논문에서는 XML 디자인 패턴 중 DTD(Document Type Definition) 모델링 시 발생하는 모호성(Ambiguity) 문제를 해결키 위한 패턴과 DOM 을 효율적으로 처리하기 위한 문서 구조 트리 운행 및 접근 패턴을 제안한다. 이 패턴들을 활용한다면 DTD 설계 및 DOM(Document Object Model) 을 처리하는 단계에서 재사용이 가능해져 개발된 기술들을 공유 가능하며 시스템 디자인 시에 발생 가능한 문제점들을 사전에 해결 할 수 있다.

1. 서론

현재 인터넷은 컴퓨터 관련 전 분야에 대해 영향을 끼치고 있으며, 이러한 인터넷의 이점을 이용하고자 여러 분야에서 활발한 연구가 이루어지고 있다. 그 중 차세대 웹 언어인 XML 의 발전과 이를 이용한 어플리케이션 제작을 위한 연구는 빠른 속도로 확산되고 있다. 이렇게 XML 을 이용한 개발 분야가 인터넷 응용 소프트웨어 개발 전반에 걸쳐 확산됨에 따라 그 규모도 점차 대형화, 복잡화 되고 있으며, 이로 인하여 XML 관련 소프트웨어 개발과 응용기술 개발에 많은 어려움을 겪고 있다. 또한 사용자들의 다양한 요구와 새로운 기술을 인터넷 환경에 적용하는 연구들이 진행 중에 있으며, 빠르게 변화하는 인터넷 환경에 발 맞추기 위한 반복적인 개발 및 유지 보수 작업이 계속적으로 진행 중에 있다.

이는 아직 XML 관련 소프트웨어 개발 분야에 체계적이며 표준화된 개발 방법이 없이 각 개발 업체가 각자 제안한 방법에 의하여 개별적인 개발 및 유지 보수가 이루어져 왔기 때문이다. 이러한 문제점들을 해결하고자 XML 개발 기술의 공유가 가능하며 수시로 변화하는 시스템 구조에 능동적으로 대처할 수 있고 재사용이 가능하도록 한 기법인 UML 과 객체지향 개념의 디자인 패턴을 XML 관련 개발 분야에 적용하는 연구가 시도되고 있다[6][7]. 그 중 디자인 패턴은 소프트웨어 설계 시 반복적으로 되풀이 하는 경험적인 설계 패턴들을 정형화하여 특정한 문제에 대한 해결 방안으로 제시된 설계의 한 방법이다. 또한, 이는 지금까지 객체지향 소프트웨어 개발에서 이루어졌던 코드나 라이브러리에 대한 재사용성이 아닌 설계 단계에서부터의 재사용성을 고려한 방법이다[4]. 이를 XML 분야에 접목한다면 DTD 모델링, 어플리케이션 개발 기술의 공유, 재사용성 고려, 분산객체 소프트웨어 개발, 체계적인 웹 시스템 설계, 유지보수 등의 문제들을 쉽게 접근 할 수 있다.

본 논문에서는 XML 디자인 패턴 적용 분야 중 DTD 모델링 시 선택적 요소와 빈도 발생자의 혼용으로 인하여 발생하는 모호성을 제거한 패턴과 DOM 을 활용한 XML 응용 소프트웨어 개발 시 객체 지향적 개발 방법으로 접근하여 재사용 가능하도록 도와주는 XML 응용 개발 디자인 패턴을 제안한다.

2. 관련 분야

XML 분야에 디자인 패턴을 적용하고 관련된 패턴들을 설계하려면 UML (Unified Modeling Language)과 디자인 패턴 분야에 대한 이해가 요구된다.

2.1. UML

OMG(Object Management Group) 에서 산업계 표준으로 제정한 UML 은 객체지향 분석(Analysis)과 설계(Design)를 위한 대표적인 모델링 언어이다[1].

UML은 확장이 용이하며 이해하기 쉽고 다양한 종류의 시스템 개발을 위한 제반 공정들의 표기법을 가지고 있다. UML 의 목적은 객체지향 시스템을 가시화, 명세화, 문서화하는 것이며 표기하려는 대상을 다이아그램을 이용하여 나타내고 그 대상에 의미를 부여한다. UML 다이아그램은 시스템이 외부에 제공하는 기능인 사용 사례들을 나타내는 사용 사례 다이아그램(Use Case Diagram), 시스템의 정적인 구조를 나타내는 클래스 다이아그램(Class Diagram), 시간에 따른 객체간의 메시지 전달을 나타내는 순차 다이아그램(Sequence Diagram), 객체의 동작을 나타내는 상태 다이아그램(State Diagram), 객체간의 협동 체계를 나타내는 협동 다이아그램(Collaboration) 등으로 구성된다[5]. UML을 이용한 요구 사항 분석과 디자인 패턴을 고려한 설계를 하게 되다면 개발중이나 개발 완료 후에 발생될 시스템 변경에 유동적으로 대처 할 수 있다

2.2. 디자인 패턴

디자인 패턴이란 시스템 설계 시에 자주 발생하는 문제들에 대한 해결책의 핵심사항을 정리한 것이다. 즉, 디자인 패턴은 객체지향 디자인 개념을 이론으로 제시하는데 그치지 않고 프로그래밍에 적용 가능하도록 구체적으로 규격화 시킨 것이다

[4]. 디자인 패턴의 장점은 객체지향 방법론의 재사용성과 모듈성을 극대화 시켜 실제 구현 과정에서 사용할 수 있게 해주는 것이다. 즉, 어떠한 특정 문제에 대한 해결책을 하나의 디자인 패턴으로 추상화 시켜 발생된 같은 문제에 대해 경험을 재사용 한다. 또한, 디자인 패턴은 공통적인 어휘를 제공함으로써 개발자들간의 의사 교환 도구로서 사용될 수 있으며 패턴을 통하여 미숙한 개발자들은 빠른 시간 내에 전문가의 경험과 기술을 이용할 수 있다.[9].

디자인 패턴은 사용 목적에 따라 세가지로 분류된다.
첫째는 생성패턴(Creational Patterns) 으로 객체의 생성과정을 추상화 하는 디자인 패턴으로 클래스 정의와 객체 생성 방식을 구조화, 캡슐화 한다. 둘째는 구조화 패턴(Structural Patterns) 으로 서로 다른 기능을 지닌 객체들이 협력을 통하여 큰 구조를 형성할 수 있도록 일반적인 구성방식을 제시한다. 셋째는 행위패턴(Behavioral Patterns) 으로 객체들의 행위를 조직화하고 관리하며 그들 사이의 의사 소통을 기술한다[4].

3. XML 디자인 패턴

XML 관련 분야에 적용 가능한 디자인 패턴은 XML DTD 모델링에 대한 패턴 부분과 XML응용 소프트웨어 개발에 대한 패턴으로 나눌 수 있다.

먼저 DTD나 스키마의 디자인과 구현에 이용되는 Information Structuring Patterns은 주로 DTD 디자인 시 일반적으로 발생하는 문제의 해결책을 찾는데 초점이 맞추어져 있다. XML 관련 응용 개발 문제를 주로 다루며 전통적인 디자인 패턴들의 합성과정제로 구현된 Program Design Patterns은 일반적인 XML 어플리케이션을 위한 공통 기본 언어와 기본 지식들을 생성한다. 이 패턴들을 이용하면 개발 프로그램에 대한 이해가 쉬워지고, 기개발된 고급 패턴 기술들을 함께 적용할 수 있어서 프로그램 품질 또한 향상시킬 수 있다.

본 논문에서는 일반적인 패턴 표기법에 의하여 XML 디자인 패턴을 표현하였다.

3.1 Ambiguous Option

이 패턴은 DTD 작성시 발생하는 문제점을 해결하기 위해 XML DTD 모델링 분야에 디자인 패턴을 적용한 것으로 선택적 내용 모델(Content Model) 구성 시 발생하는 모호성 문제를 해결하도록 도와준다.

- **Synopsis**

XML DTD를 작성할 경우 선택적 요소와 빈도 지시자가 많이 포함된 내용 모델을 구성한다면 선택적 발생 가능성과 모호성을 함께 고려하여 모델링 하여야 한다.

- **Context**

만약 책 주문을 표현하기 위한 XML DTD를 설계한다면 주문 정보의 부가적 정보 표현을 위해 주문자를 의미하는 몇 개의 Customer와 선택적 요소로서 구성되는 배달 예정일인 Date, 영수금액을 의미하는 Receipt 등이 포함된 OrderInfo라는 내용 모델을 만들 수 있다. 이러한 경우 내용 모델을 [그림1] 처럼 빈도 발생자를 사용하여 쉽게 서술할 수 있다. 그러나 이 모델은 모든 내용 요소가 선택적인 사항으로 구성이 되어 OrderInfo 요소가 발생치 않을 가능성이 존재하게 된다.

```
< !ELEMENT OrderInfo (Customer*, Date?, Receipt?) >
```

[그림1] 부가적인 주문정보에 대한 내용 모델

- **Forces**

- 여러 가지 상황에 이용될 수 있는 DTD를 만들기 위해서 많은 선택적 요소들이 DTD에 나타날 필요가 있다.
- DTD 사용한 문서 제작 시 다양한 상황에서 원하는 횟수 만큼 요소를 생성할 수 있어야 하며 또한 적은 선택의 기로에서 간단 명료하게 논리적 단위들을 사용할 수 있어야 한다.

- **Examples**

정의된 내용 모델이 발생되지 않을 수 있는 문제를 피하는 방법은 문서의 구조를 좀더 자세히 분석하여 각 내용 모델 요소들 중 하나는 반드시 발생하도록 발생 지시자를 조정하고 내용 모델을 선택적 세부 그룹으로 나누어 [그림2]와 같이 기술하는 것이다.

```
<!ELEMENT OrderInfo ((Customer+, Date?, Receipt?) | (Customer*,
Date, Receipt?) | (Customer*, Date?, Receipt?))>
```

[그림2] 그림 1의 문제를 해결한 모호한 내용 모델

그러나, 이것은 내용 모델 요소가 하나도 발생하지 않을 확률을 제거한 문제가 없는 상태로 보이지만, 이 모델은 XML 프로세서가 처리하지 못한다. 그 이유는 첫번째 요소(Customer) 정의 후 두 번째 요소(Date)가 와야 할 지에 대한 모호성 문제를 프로세스는 해결하지 못한다. XML 프로세서는 다음에 발생할 내용 요소들을 알지 못하며 여러 경우를 모두 해석하여 처리하기에는 문제가 있다. 더욱이 이런 형태의 모델은 처리한 결과를 기억하고 있다가 다시 확인하고 처리해야 하는 모델이기 때문이다.

- **Solutions**

XML DTD 모델링을 할 경우 내용 모델 구성 시 선택적 요소가 많고 빈도 지시자가 많이 부여된 내용 모델을 구성한다면 그 모델을 이용한 문서 작성시 모호성을 지닌 문서가 될 가능성이 높다. 이런 문제를 해결하기 위해서는 내용 모델을 확실하고 명확하게 정의하여 한 가지 방식으로만 해석될 수 있게 해야 하는 것이다. 즉, [그림3]과 같이 첫 선택 내용 요소의 경우 전체 발생 가능성을 고려한 모델로 빈도 지시자와 요소들을 정의하고 두 번째 선택 내용 모델에서는 첫번째에서 고려한 가능성을 제외한 형태로 정의하며 마지막 선택 요소에서는 앞의 두 선택요소에서 고려한 가능성을 제거한 상태의 내용으로 구성한다. 이것은 모델 구성에 있어서 중복인 요소 발생 가능성을 제거한 것으로 사용자에게 요소 선택에 있어서 혼란스럽지 않게 사용할 수 있도록 도와줄 뿐만 아니라 모호성을 배제한 형태로 처리 시 오류를 방지 할 수 있다.

```
<!ELEMENT OrderInfo ((Customer+, Date?, Receipt?) | (Date,
Receipt?) | (Receipt))>
```

[그림3] 모호성 문제를 해결한 모델

- **Consequences**

- DTD 설계 시 선택적 요소와 빈도 발생자의 혼용으로 인한 어려움을 해결할 수 있다.
- 선택적 내용 모델 구성 시 발생하는 모호성에 대한 문제를 해결 가능하다.

3.2. Dom Handler Pattern

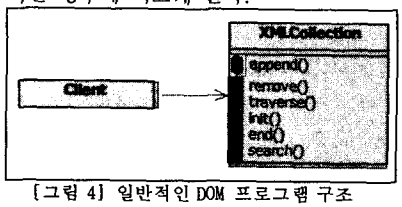
이 패턴은 XML 응용 프로그램 개발에 자주 이용되는 DOM에 관한 디자인 패턴으로 객체지향 방법을 XML 개발분야에 쉽게 접목할 수 있게 해 주는 XML 응용 단계의 패턴이다.

- **Synopsis**

DOM을 이용한 XML 응용프로그램 작성시 문서를 처리하기 위해 문서 트리 구조를 재귀 호출 방식으로 순회 운행하게 된다. 이러한 경우 DOM Handler는 문서 트리 구조에 대한 외부 노출 없이 문서 트리 구조 운행 방식을 캡슐화하고 접근 방법이나 운행 방법 변경 시 쉽게 대처할 수 있도록 해준다.

● **Context**

DOM은 HTML과 XML 문서를 위한 프로그래밍 인터페이스로 문서를 어떻게 접근하고 조작하는 가를 정의하고 있다. DOM을 이용하여 XML 응용 프로그램을 제작할 경우 오브젝트 모델에 기반한 접근 방법을 취한다. XML 문서를 읽고 파싱하여 DOM Tree 구조로 변경한 후 문서의 구조 정보나 컨텐트에 대한 여러 조작을 하기 위해 특별한 엘리먼트를 찾거나 특정의 값을 지닌 노드를 삭제나 추가하는 등의 많은 작업을 수행하게 된다. 이를 위하여 DOM Tree에서는 반복적인 트리 운행을 하게 된다. 그러나 이런 경우 특정 목적만을 위한 트리 운행 프로그래밍을 하게 되면 운행 정책이나 접근 방법에 대한 변경 시 유연하게 대처하기 어렵고 처음부터 다시 프로그래밍을 해야 하는 경우에 이르게 된다.



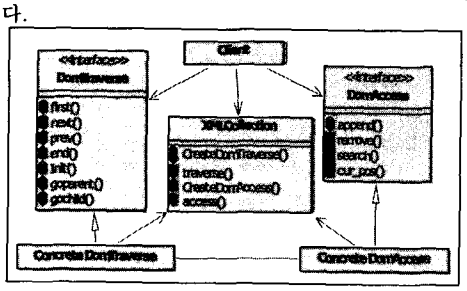
[그림 4] 일반적인 DOM 프로그램 구조

● **Forces**

- 문서 구조 접근 및 운행을 위해 문서 트리에 순회적 방문이 요구된다.
- 원하는 특정 노드에 접근한 후 문서 조작 행위가 이루어 진다.

● **Solutions**

이 문제의 해결은 DOM을 구성하는 객체 부분에서 DOM Tree를 운행하거나 접근하는 것에 대한 내용들을 따로 분리시키는 것이다.



[그림 5] DOM Handler 패턴 구조

[그림 5]처럼 운행 및 접근에 대한 방법을 DOM 처리 부분에서 따로 떼어내어 인터페이스로 구현함으로써 운행 정책이나 접근 방법에 대한 변경 시 유연한 대처를 할 수 있으며 다양한 접근 방법과 운행 정책을 쉽게 적용시킬 수 있다.

- DomTraverse : DOM Tree 구조 운행을 정의한 인터페이스
- DomAccess : DOM Tree 구조 접근을 위한 인터페이스
- Concrete DomTraverse : DomTraverse 인터페이스를 구현한 객체
- Concrete DomAccess : DomAccess 인터페이스를 구현한 객체
- XML Collection : Concrete DomTraverse, DomAccess 객체를 생성시키기 위한 역할을 수행

위 구조 패턴에 대한 코드는 다음과 같다.

```
import org.w3c.dom.*;
static Node current;
static Node cur_pos;
static Node access_pos;
public class DomIterator implements DomTraverse {
    public void first() {}     //Tree의 첫 노드로 이동
    public void next() {}      //Tree의 다음 노드로 이동
    public void prev() {}      //Tree의 이전 노드로 이동
    public void init() {}      //Tree 초기화
    public void goparent() {}  //부모 노드로 이동
    public void gochild() {}   //자식 노드로 이동
    public boolean end() {}    //마지막 노드인지 확인
    public DomIterator(Node c) { // 생성자
            current = c;
            tree.addElement(current); }
}

public class DomHandle implements DomAccess {
    public void append() {}    //노드 추가
    public void remove() {}    //노드 삭제
    public node search() {     //노드 검색
          return access_pos; }
    public Node cur_pos() {    //현재 노드 위치
          return access_pos; }
    public Node getValue() {   //현재 노드값 반환
           return current; }
     public DomHandle(Node pos) {  //생성자
          access_pos = pos; }
}
```

● **Consequences**

- DOM을 이용한 응용 프로그램 작성시 재사용성을 고려할 수 있으며 문서 트리 구조 운행 방법 변경이나 다른 운행 방법의 추가로 인하여 발생되는 문제점들을 해결해준다.
- 문서 구조 트리에 접근 할 경우 내부 구조를 외부에 노출시키지 않고 캡슐화 함으로써 더욱 안정된 프로그램 성능을 보장한다.

4. 결론

본 논문에서는 객체지향 설계에서 이용되는 디자인 패턴을 이용하여 XML 개발 분야에 적용 가능한 두 가지 디자인 패턴을 제안하였다. Ambiguous Option 패턴은 선택적 반복 요소와 빈도 발생자의 혼용으로 인한 DTD 설계의 어려움과 모호성 문제를 해결할 수 있다. DOM Handler 패턴은 XML 개발에 자주 이용되는 DOM의 운행 및 접근 부분을 분리시켜 객체 지향 방법을 XML 분야에 적용할 수 있도록 해주며 유지보수나 재개발 시에 재사용할 수 있다. 구현 중심의 XML 분야에 디자인 패턴 개념을 적용하면 설계 단계에서의 재사용성과 개발 기술의 공유로 기술 발전 속도를 촉진시키고 개발자 상호간 이해를 증진시킬 수 있다. 또한, 향후 XML 응용 분야가 급증하는 만큼 XML 디자인 패턴 분야는 중요성이 크게 부각될 것이며 패턴 연구는 XML을 다양한 분야에 쉽게 적용하고 고급 기술을 사용하도록 이끌어 주는 기틀이 될 것이다. 향후 연구로는 객체 개념을 XML 웹 어플리케이션 개발 분석 단계에서부터 적용하여 XML 분석 패턴을 추출하고 이를 이용하여 디자인 패턴 및 프레임 워크를 설계하는 것이다.

5. 참고문헌

[1] OMG, "*UML Specification Version 1.3*", Object Management Group, January 1999.
[2] Document Object Model (DOM) Level 1 Specification, http://www.w3.org/TR/REC-DOM-Level-1/
[3] Craig Larman, *Applying UML and Patterns*, 1998.
[4] Erich G amma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
[5] Grady Booch, *The Unified Modeling Language User Guide*, Addison Wesley, 1998.
[6] Rick Jelliffe, 1998, *The XML & SGML Cookbook: Recipes for Structured Information*, Charles F. Goldfarb Series on Open Information Management
[7] W.Eliot Kimber, "*Using UML To Define XML Document Types*", OASIS XML Articles and Papers, Jan 2000.
[8] Neil Bradley, "*The XML Companion*", Addison Wesley, 2000.
[9] 애플리케이션의 보안서비스 구현을 위한 디자인 패턴 : Folder / 이진영, 1997, 한국과학기술원