DTD의 분석을 통한 XML 문서 저장

신병주, 진 민
경남대학교 컴퓨터공학과

Storing XML documents through analyzing DTD

Byung-Joo Shin, Min Jin
Dept. of Computer Engineering, Kyungnam University
E-mail : challenger@hawk.com.kyungnam.ac.kr, mjin@hawk.com.kyungnam.ac.kr

요 약

XML은 인터넷 기반의 비즈니스 환경에서 데이터 교환의 표준으로 확고한 위치를 확보하였다. 따라서, XML로 표현된 비즈니스 데이터를 가장 보편적인 DBMS인 관계 데이터베이스에 저장하기 위한 요구가 증가하고 있다. 그러나, XML과 관계 데이터베이스 간의 구조적 불일치에 의해 발생하는 문제점들을 해결하기 위해 XML 문서를 관계 데이터베이스에 저장하기 위한 별도의 저장 방법에 대한 연구가 활발히 진행되었다. 그 중의 한 방법이 DTD의 분석을 통해서 저장 방법을 결정하는 것이었다. 그러나 DTD는 XML과 다른 문법 구조를 갖기 때문에 파싱의 어려움이 존재하고, 관계 데이터베이스와의 원활한 연계가 어려우며, 사용자가 원하는 형태를 정의해서 사용할 수가 없기 때문에 유연성이 떨어지는 단점을 가지고 있다. 따라서, 본 논문에서는 DTD를 통한 XML 문서의 저장 방법에서 DTD로부터 추출한 저장 구조를 XML 형태의 문서로 변환하여 저장할 수 있는 방법을 제안한다.

1. 서론

XML은 문서의 내용에 대한 정보를 전달하기 위해 의미있는 태그를 사용자가 정의해서 사용할 수 있다. 또한, XML은 문서에 대한 구조 정보를 제공할 뿐만 아니라, XML 태그는 데이터를 해석하는데 사용할 수도 있다. 이와 같은 XML의 장점은 인터넷 기반의 비즈니스 환경에서 XML이 데이터 교환의 표준으로 확고한 위치를 차지할 수 있게 하였다. 그리고 대부분의 비즈니스 관련 데이터는 현재 관계 데이터베이스에 저장되어져 있다. 따라서, XML 문서를 관계 데이터베이스에 저장하기 위한 요구가 증가하고 있다. 그러나 관계 데이터베이스와 달리 XML은 데이터 간의 계층화된 구조를 가지고, 관계 데이터베이스의 테이블과 컬럼의 구조와 XML의 엘리먼트, 애트리뷰트의 구조가 일치하지 않는다. 또한, 관계 데이터베이스의 각 데이터 인스턴스들이 데이터와 독립적인 스키마를 가지는 반면에 XML의 데이터는 스키마가 태그로서 데이터와 함께 표현되어 진다[3]. 이과 같은 구조적 차이점들로 인해 XML 문서를 관계 데이터베이스에 저장하기에는 어려운 점이 존재하였고 이 문제점들을 해결하기 위하여 많은 연구가 진행되어 왔다. 그 중의 한 방법이 DTD 그래프를 통하여 DTD를 분석하여 저장 구조를 결정하는 방법이다[6]. 그러나, DTD는 XML과 다른 문법 구조를 갖기 때문에 DTD의 파싱을 위한 별도의 파서가 필요하게 되고 다양한 데이터 타입을 정의할 수 없기 때문에 데이터베이스와의 원활한 연계가 어렵다. 그리고 사용자가 원하는 형태를 정의해서 사용할 수 없기 때문에 유연성이 떨어지는 단점을 가지고 있다.

따라서, 본 논문에서는 XML 문서를 관계 데이터베이스에 저장함에 있어 DTD로부터 저장 구조에 대한 정보를 추출하여 XML과 같은 문법 형태의 문서로 변환하여 저장하는 방법을 제안하였다. 즉, XML 데이터의 관계 스키마 사상을 위한 중간 형태의 XML 파일을 생성해서 이 파일을 통해 관계 데이터베이스에 효과적으로 XML 문서를 저장하도록 함으로써

본 연구는 경남대학교 학술연구조성비에 의해 연구되었음.

DTD가 가지는 단점을 보완하고자 하였다.

2. DTD 그래프에 의한 관계 스키마의 생성

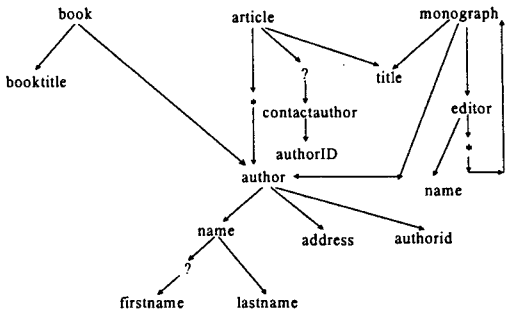
```
<!ELEMENT book (booktitle, author)>
<!ELEMENT booktitle (#PCDATA)>
<!ELEMENT article (title, author*, contactauthor)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID IDREF IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor (monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (name, address)>
<!ATTLIST authorid ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address ANY>
```

<그림 1> DTD 예

관계 스키마는 개체-관계 모델과 같은 데이터 모델로부터 생성되어진다. 개체-관계 모델에서는 개체와 그 개체의 특징을 기술해 주는 속성 간의 명확한 구분이 있기 때문에 관계 스키마 변환시 개체는 릴레이션으로, 속성은 릴레이션의 속성으로 사상되어진다. 이처럼 XML DTD로부터 관계 스키마로 사상하는 기본적인 방법은 DTD의 각 엘리먼트는 관계 스키마의 릴레이션으로, 엘리먼트의 애트리뷰트는 릴레이션의 속성으로 사상하는 방법이다. 그러나, XML 문서의 엘리먼트, 애트리뷰트의 구조와 관계 스키마의 릴레이션, 속성의 구조는 항상 일치하는 것이 아니다. 즉, XML 문서에서 엘리먼트로 선언되어진 것이 관계 스키마에서 릴레이션의 속성으로 사상되어 지는 경우가 많다. 이와 같은 단순한 XML DTD의 관계 스키마 변환 방법은 릴레이션의 대량 생산에 의한 저장 공간의 낭비와 관계 데이터베이스에 저장된 XML 데이터에 대한 질의 처리시 효율성을 저하시키는 문제가 발생하게 된다[6].

이와 같이, XML DTD와 관계 스키마의 구조적 불일치에서 발생하는 저장 방법의 문제를 해결하기 위한 방법으로 DTD 그래프에 의한 관계 스키마 생성 방법이 있다.

<그림 2>는 <그림 1>의 DTD를 바탕으로 생성되어진 DTD 그래프이다. 즉, DTD를 분석한 DTD 그래프 상에서 In-Degree의 값에 따라 사상 방법을 결정하는 것이다[6]. <그림 2>의 DTD 그래프에서 book



<그림 2> DTD 그래프

엘리먼트처럼 in-degree 값이 0인 경우에는 릴레이션으로, booktitle 엘리먼트와 같이 in-degree가 1인 경우에는 릴레이션의 속성으로 사상한다. 한편, in-degree가 2이상일 경우에는 별도의 릴레이션을 생성하는 것을 원칙으로 하나, 상위 엘리먼트와 하위 엘리먼트의 상관관계 분석을 통해 독립적인 관계(예: auther 엘리먼트)라면 릴레이션으로, 종속적인 관계(예: name 엘리먼트)라면 릴레이션의 속성으로 표현하도록 함으로써 질의 처리의 효율성을 향상시킬 수 있도록 한다. 이 때, 엘리먼트가 단말 노드(예: title 엘리먼트)이거나 다중값을 갖는 엘리먼트, 순환 구조를 갖는 엘리먼트(예: monograph 엘리먼트) 등은 in-degree의 값과는 무관하게 별도의 릴레이션을 생성한다[11][12].

복수 개의 상위 엘리먼트를 가지거나 다중값을 갖는 엘리먼트, 순환 구조를 갖는 엘리먼트 등은 다른 릴레이션과의 관계가 표현되어져야 한다. 또한, 엘리먼트가 릴레이션의 속성으로 사상됨으로써 발생하는 문제의 해결이 요구되어 진다. 이와 같은 문제의 해결을 위해 관계 스키마의 생성시 별도의 속성들을 추가한다[6][11][12].

DTD 그래프의 in-degree 값에 의한 저장 방법 외에도 추가적인 저장 방법이 요구되는 경우가 있다. 그 중의 하나는 Mixed 엘리먼트와 Any 엘리먼트의 저장 방법이다. Mixed 엘리먼트와 Any 엘리먼트는 하위 엘리먼트로 다양한 형태의 엘리먼트가 순서에 관계없이 존재할 수 있으므로 관계 데이터베이스에 저장하기 위해서는 복잡한 처리과정이 필요하게 된다. 또한 ID와 IDREF(S) 간에는 다대다의 관계로서 값을 참조하는 경우가 발생한다. 이와 같은 다대다의 관계를 관계 데이터베이스에 표현하기 위해서도 추가적인 저장 방법이 요구되어 진다[4][7].

그리고 XML 문서가 관계 데이터베이스에 분리되어 저장됨으로써 원래의 XML 문서가 가진 구조정보가 손실된다. 따라서, 원래의 XML 문서가 가진 구조정보를 별도로 저장함으로써 XML 문서의 복원을 원활하게 하고, 질의 처리시 데이터베이스에 대한 접근 횟수를 감소하여 효율성을 향상시킨다[11][12].

3. DTD 분석을 통한 매핑 파일의 생성

DTD는 XML에 대한 규칙을 정의함에 있어 XML과 다른 문법을 사용하기 때문에 XML 문서와 일관성이 없고 데이터 타입에 제한적이기 때문에 데이터베이스에 원활한 연계가 어려우며, 사용자가 원하는 형태를 정의해서 사용할 수가 없어 유연성이 떨어진다. 따라서, DTD의 분석을 통하여 추출한 저장 구조의 정보를 아래와 같은 규칙에 의해 XML 형태로 변환한 후 XML 문서와 함께 DOM 트리를 생성하여 저장한다[5][8][9].

```
<!ELEMENT XMLToRDBMS (IgnoreRoot*, RelationMap+)>
<!ELEMENT IgnoreRoot (#PCDATA)>
<!ELEMENT RelationMap
          ((ToTable | ToSpecialTable), AttributeMap+)>
<!ATTLIST RelationMap
          ElementName CDATA #REQUIRED>
```

XMLToRDBMS 엘리먼트는 매핑 파일의 루트 엘리먼트이다. *IgnoreRoot* 엘리먼트는 XML 문서의 루트로 정의되어진 엘리먼트로 실질적인 데이터 없이 구조 정보만을 갖는 엘리먼트를 위해 별도로 릴레이션을 생성할 필요가 없는 경우에 사용되는 엘리먼트이다. 그리고 *RelationMap* 엘리먼트는 생성되어질 릴레이션에 대한 데이터를 표현하게 된다. *RelationMap* 엘리먼트의 *ElementName* 애트리뷰트는 관계 스키마의 릴레이션으로 사상되어지는 DTD 상의 엘리먼트 명을 가리킨다.

```
<!ELEMENT ToTable (PrimaryKey, ParentID?)>
<!ATTLIST ToTable
          TableName CDATA #REQUIRED>
<!ELEMENT PrimaryKey EMPTY>
<!ATTLIST PrimaryKey (PCDATA)>
          Generate (Yes | No) #REQUIRED
          isRoot (Yes | No) #REQUIRED>
<!ELEMENT ParentID (ParentCode?, ParentElement+)>
<!ATTLIST ParentID
          ColumnName CDATA #REQUIRED>
<!ELEMENT ParentCode (PCDATA)>
<!ELEMENT ParentElement (PCDATA)>
<!ATTLIST ParentElement
          ForeignKey CDATA #REQUIRED>
```

ToTable 엘리먼트는 릴레이션으로 사상되는 엘리먼트가 실제로 데이터베이스에 생성되어지는 방법과 추가적인 속성의 생성과 관련한 정보를 포함하고 있다. *TableName* 애트리뷰트는 데이터베이스에 생성되어지는 테이블 명을 나타낸다.

그리고 *PrimaryKey* 엘리먼트는 기본키의 역할을 할 컬럼의 생성 여부에 대한 정보를 갖고 있다. 즉, *Generate* 애트리뷰트가 *Yes* 값을 가지면 별도로 기본키의 역할을 하는 컬럼을 생성하게 되고 *No* 값을 가지는 경우에는 기존의 컬럼 중 기본키의 역할을 하는 컬럼을 지정하게 된다. 그리고 *isRoot* 애트리뷰트는 특정 엘리먼트를 릴레이션의 속성으로 사상했을 경우 이 엘리먼트가 XML 문서의 루트가 되는 경우에 데이터 검색 시 문제가 발생하게 되는 문제점을 해결하기 위해 존재하는 애트리뷰트이다.

ParentID 엘리먼트는 다른 릴레이션과의 관계를 표현하는 엘리먼트로서 외래키의 역할을 하는 컬럼명을 *ColumnName* 애트리뷰트에 나타낸다.

ParentCode 엘리먼트와 *ParentElement* 엘리먼트는 복수 개의 상위 엘리먼트를 갖는 엘리먼트, 순환 구조를 갖는 엘리먼트를 표현하기 위해 필요하다. *ParentCode* 엘리먼트는 데이터베이스에 생성되는 컬럼명을 나타내고 *ParentElement*는 상위 엘리먼트의 명을 가리키며 *ParentElement*의 *ForeignKey* 애트리뷰트는 상위 엘리먼트의 속성들 중 관계를 갖는 속성명을 표현하게 된다.

```
<!ELEMENT ToSpecialTable (SubElement, RelationMap+)>
<!ATTLIST ToSpecialTable
          TableName CDATA #REQUIRED
          PrimaryKey CDATA #REQUIRED>
<!ELEMENT SubElement
          (ForeignKey, TableName, LookupKey, Sequence)>
<!ATTLIST SubElement
          TableName CDATA #REQUIRED>
<!ELEMENT ForeignKey (PCDATA)>
<!ELEMENT TableName (PCDATA)>
<!ELEMENT LookupKey (PCDATA)>
<!ELEMENT Sequence (PCDATA)>
```

ToSpecialTable 엘리먼트는 별도의 저장방법이 요구되는 Mixed element와 Any element의 저장을 위한 정보를 포함하고 있다. *SubElement* 엘리먼트는 Mixed element와 Any element의 하위 엘리먼트들에 대한 정보를 저장하기 위한 엘리먼트이다. *ForeignKey* 엘리먼트는 Mixed element와 Any

element와의 관계 표현을 위한 컬럼, *TableName* 엘리먼트는 하위 엘리먼트의 명을 저장하기 위한 컬럼, *LookupKey* 엘리먼트는 하위 엘리먼트의 특정 열을 가리키기 위한 컬럼, *Sequence* 엘리먼트는 순서 정보를 저장하기 위한 컬럼에 대한 정보를 가지고 있다. 그리고 하위 엘리먼트들의 저장을 위해 별도의 릴레이션을 생성하고 있다.

```
<!ELEMENT AttributeMap
        (IgnoreElement | ToColumn | ToSeperateTable)>
<!ELEMENT IgnoreElement EMPTY>
<!ATTLIST IgnoreElement
        ElementName CDATA #REQUIRED
        isRoot (Yes | No) #REQUIRED>
<!ELEMENT ToColumn (ColumnName, AttributeRef*)>
<!ATTLIST ToColumn
        AttributeName CDATA #REQUIRED>
<!ELEMENT ColumnName (PCDATA)>
<!ATTLIST ColumnName
        isRoot (Yes | No) #REQUIRED>
<!ELEMENT AttributeRef (PCDATA)>
<!ATTLIST AttributeRef
        RefTable CDATA #REQUIRED>
<!ELEMENT ToSeperateTable (RelationMap)>
```

AttributeMap 엘리먼트는 테이블의 컬럼으로 사상되는 엘리먼트와 애트리뷰트들의 정보를 포함하고 있다. *IgnoreElement*는 테이블의 컬럼으로 사상되어지는 엘리먼트이지만 실질적인 데이터 없이 구조적인 정보만을 갖고 있는 엘리먼트를 나타낸다.

ToColumn 엘리먼트는 실제로 데이터베이스에 테이블의 컬럼으로 사상되어지는 엘리먼트와 애트리뷰트들의 정보를 가리킨다. *AttributeName* 애트리뷰트는 XML 문서에서 관계 스키마의 속성으로 사상되어지는 엘리먼트와 애트리뷰트의 명을 나타내고 *ColumnName* 엘리먼트는 실제 생성되어지는 칼럼명을 나타낸다.

AttributeRef 엘리먼트는 XML 문서에서 ID와 IDREF(S)의 관계처럼 다른 테이블의 값을 참조할 경우를 표현한다. *AttributeRef* 엘리먼트는 참조하는 속성 값을 나타내고 *AttributeRef* 엘리먼트의 *RefTable* 애트리뷰트는 참조하는 테이블을 가리킨다.

ToSeperateTable 엘리먼트는 테이블의 컬럼으로 사상되어져 다중값을 갖는 경우 별도의 테이블을 생성할 수 있도록 하는 엘리먼트이다.

4. DTD 분석을 통한 매핑 파일의 생성 예

<그림 3>은 <그림 2>의 DTD 그래프에서 추출한 저장 구조의 정보를 매핑 파일로 변환한 예이다.

```
<XMLToRDBMS>
  <RelationMap ElementName="book">
    <ToTable TableName="book">
      <PrimaryKey Generate="Yes" isRoot="No">bookid</PrimaryKey>
    </ToTable>
    <AttributeMap>
      <ToColumn AttributeName="booktitle">
        <ColumnName isRoot="Yes">booktitle</ColumnName>
      </ToColumn>
    </AttributeMap>
  </RelationMap>
  <RelationMap ElementName="article">
    <ToTable TableName="article">
      <PrimaryKey Generate="Yes" isRoot="No">articleid</PrimaryKey>
    </ToTable>
    <AttributeMap>
      <IgnoreElement ElementName="contactauthor" isRoot="Yes">
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="authorID">
        <ColumnName isRoot="No">authorID</ColumnName>
        <AttributeRef RefTable="author">authorid</AttributeRef>
      </ToColum>
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="title">
        <ColumnName isRoot="Yes">title</ColumnName>
      </ToColumn>
    </AttributeMap>
  </RelationMap>
  <RelationMap ElementName="author">
    <ToTable Tablename="author">
      <PrimaryKey Generate="No" isRoot="No">authorid</PrimaryKey>
      <ParentID ColumnName="ParentID">
        <ParentCode>ParentCode</ParentCode>
        <ParentElement ForeignKey="bookid">book</ParentElement>
        <ParentElement ForeignKey="articleid">article</ParentElement>
        <ParentElement
                ForeignKey="monographid">monograph</ParentElement>
      </ParentID>
    </ToTable>
    <AttributeMap>
      <IgnoreElement ElementName="name" isRoot="Yes">
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="firstname">
        <ColumnName isRoot="Yes">firstname</ColumnName>
      </ToColumn>
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="lastname">
        <ColumnName isRoot="Yes">lastname</ColumnName>
      </ToColumn>
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="authorid">
        <ColumnName isRoot="No">authorid</ColumnName>
      </ToColumn>
    </AttributeMap>
  </RelationMap>
  <RelationMap ElementName="monograph">
    <ToTable Tablename="monograph">
      <PrimaryKey Generate="Yes" isRoot="Yes">monographid</PrimaryKey>
      <ParentID ColumnName="ParentID">
        <ParentCode>ParentCode</ParentCode>
        <ParentElement
                ForeignKey="monographid">monograph</ParentElement>
      </ParentID>
    </ToTable>
```

```
    <AttributeMap>
      <IgnoreElement ElementName="editor" isRoot="Yes">
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="name">
        <ColumnName isRoot="Yes">name</ColumnName>
      </ToColumn>
    </AttributeMap>
    <AttributeMap>
      <ToColumn AttributeName="title">
        <ColumnName isRoot="Yes">title</ColumnName>
      </ToColumn>
    </AttributeMap>
  </RelationMap>
</XMLToRDBMS>
```

<그림 3> 매핑 파일의 예

5. 결론

XML 문서의 구조와 관계 데이터베이스의 구조의 불일치로 인해 XML 문서를 관계 데이터베이스에 저장하기 위해서는 어려운 점들이 존재한다. 따라서, DTD 그래프의 in-degree의 값에 따라 관계 스키마를 생성하도록 하였다. 그러나, DTD는 다양한 데이터 타입을 정의할 수 없기 때문에 데이터베이스와 원활한 저장이 어렵고 사용자가 원하는 형태를 정의하여 사용할 수 없기 때문에 유연성이 떨어지는 단점을 가지고 있다. 또한, DTD는 XML에 대한 규칙을 정의함에 있어 XML과 다른 문법을 사용하기 때문에 XML 문서와 일관성을 유지하기가 어렵다. 따라서, 본 논문에서는 DTD 그래프의 분석을 통하여 관계 데이터베이스에 XML 문서를 효율적으로 저장할 수 있는 저장 구조에 대한 정보를 추출하여 XML 문서 형태의 매핑 파일을 생성하도록 하였다. 즉, 저장 구조에 대한 정보를 가진 매핑 파일과 저장될 XML 문서로부터 생성되는 DOM 트리에 의해 XML 문서를 관계 데이터베이스에 효율적으로 저장할 수 있도록 하였다. 이는 사용자가 XML 문서의 저장을 위한 관계 데이터베이스의 스키마 정보를 정의할 수 있도록 하고 데이터베이스와의 연계에서도 유연한 구조를 가질 수 있도록 하였다. 또한, 매핑 파일의 규칙을 정의함에 있어 기존의 XML 문서 저장시스템에서 지원이 부족했던 다중값을 갖는 엘리먼트, 복수 개의 상위 엘리먼트를 갖는 엘리먼트, 순환 구조를 갖는 엘리먼트 등의 저장을 위한 규칙을 정의함으로써 XML 문서가 갖는 장점을 최대한 활용할 수 있도록 하였다.

[참고문헌]

[1] D. Florescu, D. Kossmann, "Storing and Querying XML Data using an RDBMS", Data Engineering Bulletin, Vol. 22, No. 3, 1999

[2] D. Kroenke, *Database Processing*, Prentice Hall, 2000

[3] E. Xavier. P, "Using Extensible Query Language(XQL) for Database Applications, Proceedings of the Eighth Annual IEEE International Conference, 2001

[4] F. Boumphrey, O. Direnzo, J. Duckett, J. Graf, P. Houle, D. Hollander, T. Jenkins, P. Jones, A. Kingsley-Hughes, *Professional XML Applications*, Worx Press, 1999

[5] H. Maruyama, K. Tamura, N. Uramoto, *XML and Java Developing Web Applications*, Addison-Wesley, 1999

[6] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton, "Relational Databases for Querying XML Documents : Limitations and Opportunities", VLDB, Edinburg, Scotland, 1999

[7] K. Williams, M. Brundage, P. Dengler, J. Gabriel, A. Hoskinson, M. Kay, T. Maxwell, M. Ochoa, J. Papa, M. Vanmane, *Professional XML Databases*, Wrox Press, 2000

[8] R. Bourret, "Mapping DTDs to Databases", www.xml.com/pub/a/2001/05/09/dtdtodbs.html, 2001

[9] R. Bourret, C. Bornhovd, A. Buchmann, "A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Dtabases", TR-DBS99-1, DVS, Dep.CS, Darmstadt U. of Technology, Germany, Dec. 1999

[10] T. Bray, J. Paoli, C. M. Sperberg-McQueen, "Extensible Markup Language(XML) 1.0", www.w3.org/TR/REC-xml

[11] 신병주, 진민, 정민수, "XML 문서의 관계 데이터베이스 저장", 한국정보처리학회 추계학술발표논문집 제8권 제2호, pp.55-58, 2001

[12] 신병주, 진민, "XML 구성요소의 릴레이션으로의 변환", 한국정보처리학회 추계학술발표논문집 제8권 제2호, pp.35-38, 2001