사용자 정보에 기반한 XML문서 전달 시스템
(A Personalized XML Documents Delivery System)

유 상 원 † 이 형 동 † 김 형 주 ††
(Sang Won Yoo) (Hyung Dong Lee) (Hyoung-Joo Kim)

요 약 그동안 메일이나 뉴스등을 필터링하여 사용자에게 전달해 주는 많은 시스템들이 개발되었다. 이 시스템들이 필터링하는 문서들은 일반 텍스트나 HTML으로 작성된 것들이었다. 새로운 정보 교환 표준으로 떠오르고 있는 XML은 문서를 필터링하는 시스템들에 있어서도 다른 접근 방법을 요구하고 있다. 본 논문에서 구현한 시스템은 XML이 가진 스키마 표현 능력과 구조 정보를 이용하여 사용자 정보를 기술하는 방법을 제안한다. 사용자 정보는 DTD로 부터 추출된 정보를 이용하여 DTD를 따르는 문서내의 특정 부분을 가리킬 수 있도록 만들어진다. 또한 기존의 필터링 시스템이 문서단위의 필터링에 초점을 맞추고 있는 것을 개선하기 위해 문서에서 사용자가 원하는 부분만을 제공하고 있다. 사용자 정보는 XML질의에 반영되어 XML로 이루어진 문서에서 일부분을 얻어내는데 사용된다.
키워드 : XML, 개인화, SDI 시스템

Abstract There have been many filtering systems covering mail or news. Documents filtered by them consist of general text or HTML. XML is emerging as a new standard for informatin exchange. So, filtering systems need new approaches in dealing with xml documents. Our system suggests a method to describe user profiles with XML's ability to represent schema and structure. An user profile is made from DTD information and it is supposed to point the specific part of a document conforming to the DTD. More, it is different from the existing systems in extracting part of a document. An user profile is reflected in XML query to get part of an XML document.
Key words : XML, Personalization, SDI

1. 서 론

인터넷(internet), 인트라넷(intranet) 등 네트워크의 활성화와 더불어 온라인 정보가 넘쳐나고 있다. 기존의 온라인 정보들은 대부분이 HTML이나 텍스트로 이루어져 있었으나 최근들어 XML(eXtensible Markup Language)이 특정 분야의 데이타를 표현하거나 기업간의 정보교환 목적으로 널리 사용되고 있다. 또한 XML을 지원하기 위한 여러 응용프로그램들이 발표되고 있으며 이미 IBM이나 Oracle, Microsoft와 같은 대기업들도 XML을 지원하는 제품들을 내놓고 있다.

이렇게 XML에 관한 관심이 고조되는 이유는 XML이 문서의 구조를 표현할 수있는 능력을 가지고 있다는 데 있다. 예를 들어 신문을 XML로 기술하면 태그정보만 가지고 어떤 부분부터 어떤 부분까지가 섹션이며 또 어떤 부분이 기사를 기술하는지에 관한 정보들을 쉽게 표현할 수 있다. 또한 HTML과는 달리 태그를 자유롭게 정의하여 사용할 수 있기 때문에 특정 분야에 맞는 문서들을 얼마든지 만들어 낼 수 있다.

이러한 XML의 장점에도 불구하고 Information Filtering 분야나 SDI(Selective Dissemination of Information)환경에서는 XML에 대한 응용이 이루어지지 않고 있는 실정이다. 일반적으로 사용자에게 적절한 정보를 제공해주는 필터링 시스템들은 사용자 정보(user profile)에 기반하여 해당 정보가 사용자에게 적합한지를 판별한다. 기존의 시스템들은 HTML이나 텍스트로 이루어진 문서들을 필터링하여 사용자에게 전달하기 때문에 사용자 정보를 키워드의 집합으로 표현한다. 그리고 해당문서에 나타나는 키워드들과 사용자 정보를 비교하는 과정을 거쳐 문서가 전달된다. 하지만 필터링하여야 할 대상이 XML문서인 경우 기존의 접근방법과 같은 방법을 적용하면 XML이 구조 정보를 가지고 있다는

· 본 논문은 BK21 사업 및 ITRC 사업의 지원을 받았음
† 학생회원 : 서울대학교 컴퓨터공학부
swyoo@oopsla.snu.ac.kr
hdlee@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2001년 11월 23일
심사완료 : 2003년 5월 20일

장점을 살릴 수 없게 된다.

예를 들어 사용자에게 정기적으로 신문(e-newspaper)을 제공하는 시스템이 있다고 가정하자. 이때 사용자가 신문 전체가 아니라 야구와 관련된 기사만을 전달받고 싶다면 문서내 경로 정보(path information)를 이용한 다음과 같은 질의를 통해 손쉽게 사용자가 원하는 부분만을 얻어낼 수 있다[1,2].

SELECT X FROM Newspaper.Sports.Baseball X

문서내의 경로정보가 곧 문서의 구조 정보이므로 사용자 정보를 문서내 경로정보를 이용하여 표현한다면 위에서 언급한 질의어 등을 통해 사용자가 원하는 정보만을 쉽게 추출하여 전달할 수 있을 것이다. 또한 여러 개의 XML문서의 내용을 모아서 전달하는 일도 가능하다.

XML이 가진 또 하나의 장점은 데이타를 표현하는 부분과 프레젠테이션 부분이 분리되어 있다는 점이다. 따라서 하나의 문서가 여러가지 다른 형태로 제공될 수 있다. 이때 필터링의 단위가 문서가 아니라 문서내의 일부분을 추출하는 것이라면 XML문서를 다른 형태의 포맷으로 변환하는 과정에서도 문서의 구조 정보가 필요하게 된다. 즉 문서의 일부분만을 다른 형태로 변환하여 제공해야 하는 것이다.

우리가 구현한 시스템은 XML문서들이 다수의 사용자들에게 제공되는 환경에서 사용자가 원하는 내용만 골라 재구성하여 제공하는 시스템이다. 이때 위에서 언급한 XML의 두가지 특징을 이용하여 사용자가 원하는 정보만을 제공한다. 먼저 사용자 정보는 문서내의 경로정보를 표현하기 위해 XPath[3]의 형태로 표현된다. 제공하려는 문서들은 신문이나 잡지 등을 위한 DTD(Document Type Definition)를 따른다고 가정하고 DTD로부터 구조 정보를 추출하여 사용자 정보와 대응시키는 방법을 제안하였다. (DTD에 대한 설명은 2장에서 언급한다.) 또 DTD에서 얻어낸 구조 정보를 바탕으로 문서를 제공하는 제공자가 사용자가 원하는 부분을 선택하는 단위를 결정하도록 해준다. 예를 들면 신문에서 사용자가 원하는 내용을 섹션단위로 제공받게 할 것인지 기사단위로 제공받게 할 것인지 문서 제공자가 결정할 수 있다. 그리고 현재 모든 웹 브라우저에서 HTML을 지원하므로 사용자에게 제공하는 문서는 XSLT[4]를 이용하여 HTML로 변환한 후 제공한다. 이때 사용자 정보를 이용하여 문서내에서 사용자가 원하는 부분을 복수의 문서에서 추출하여 제공하는 기법을 제안하였다.

본 논문의 구성은 다음과 같다. 2장에서는 DTD와 데이타 모델 및 XPath, XSLT에 관해 설명하고 3장에서는 관련연구에 대해 설명한다. 4장에서는 구현한 시스템의 전체적인 구조와 DTD로부터 사용자 정보를 기술하기 위한 문서내의 구조정보를 추출하는 방법, 사용자 정보를 바탕으로 각 사용자에게 서로 다른 문서를 제공하는 방법에 대해 설명한다. 5장에서는 마지막으로 결론 및 향후 연구 방향에 대해 설명한다.

2. 배경지식

2.1 XML과 DTD(Document Type Definitions)

XML은 그 문서의 구조를 나타내는 DTD와 실제 XML 데이타로 이루어지는데 XML 데이타는 엘리먼트(element)라는 항목이 연속된 형태로 구성되고, 중첩될 수 있다. 또한 각 엘리먼트는 애트리뷰트(attribute)를 가질 수 있다. 이러한 XML은 한 엘리먼트가 다른 엘리먼트를 참조하는 경우를 배제하면 트리 형태의 구조로 대응시킬 수 있다. XML문서 내에 포함된 엘리먼트들은 하나의 노드(node)로 표현이 되고 노드간의 에지(edge)들은 부모자식 관계를 나타내게 된다. 이때 노드들은 순서를 가지는 리스트이다. 일반 애트리뷰트들은 [5]의 방법으로 애트리뷰트를 가지지 않는 XML로 변환 가능하므로 데이타 모델에 영향을 미치지 않는다.

XML을 루트를 가진 트리 형태의 구조로 가정하면 XML의 엘리먼트들은 각각 부모 자식 관계를 가지게 되고 전체적으로 계층적인 구조를 가지게 되므로 이러한 관계에 의해 XML문서의 내용을 일정한 단위로 분할할 수 있다.

이와 같은 엘리먼트간의 관계를 추출해 내기 위해 DTD가 필요하다. DTD는 이를 따르는 XML문서의 문법(grammar)역할을 한다고 할 수 있으며 동시에 XML문서에 의해 표현되는 데이타들의 스키마(schema)정보를 나타낸다고 할 수 있다[6]. 즉 DTD를 이용하면 이를 따르는 XML문서의 경우 문서를 읽기 전에 먼저 전체적인 XML문서의 구조를 알아낼 수가 있다. DTD는 각 데이타 요소들의 관계를 정규식(regular expression)을 이용하여 표현하고 있는데 그 예를 살펴보면 그림 1과 같다.

그림 1에서 간단한 DTD와 이 DTD에 맞춰 기술된 XML을 보이고 있다. 그림 1의 왼쪽에 나타난 DTD를 보면 우리는 다음의 정보를 얻을 수 있다.

1. ‘newspaper’는 ‘economy’와 ‘sports’의 두 가지 섹션으로 이루어질 수 있다.
2. ‘sports’ 섹션은 ‘baseball’과 ‘football’의 순서로 두 가지 하위섹션을 가진다.
3. ‘baseball’이 존재할 경우 경로정보는 “newspaper/sports/baseball”로 주어진다.

이러한 DTD가 가지는 구조정보들을 이용하여 문서 제공자가 원하는 단위대로 문서를 분할한다. 그리고 사용자가 원하는 단위들을 선택할 때 사용자 정보가 문서

```
<! DOCTYPE newspaper[
<! ELEMENT newspaper(economy?,sports?)>
<! ELEMENT economy(company*,stock)>
<! ELEMENT company(#PCDATA)>
<! ELEMENT stock(#PCDATA)>
<! ELEMENT sports(baseball,football)>
<! ELEMENT baseball(#PCDATA)>
<! ELEMENT football(#PCDATA)>
]>
```

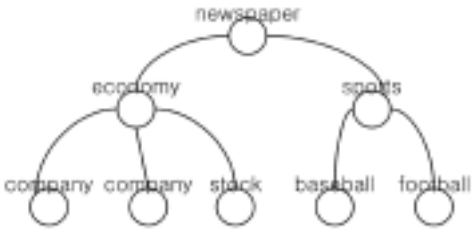


그림 1 DTD와 DTD를 따르는 XML

내의 구조 정보로 대응되도록 한다.

2.2 XSLT(XSL Transformation language)와 XPath

XML문서는 외형정보와 독립적이기 때문에 쉽게 읽을 수 있는 프레젠테이션 계층이 필요하다. 본 논문에서는 XSLT와 XPath를 이용하여 XML문서를 HTML문서로 변환한 후 사용자에게 보여지도록 하였다.

XSLT는 다른 XML질의어들과 마찬가지로 질의의 결과를 새로운 그래프형태로 만들어주는 기능(restructuring capability)을 가지고 있다[1,2,7]. 이를 이용하면 XML문서를 사용자가 쉽게 읽을 수 있는 HTML이나 다른 포맷으로 변환 가능하다.

XSLT는 여러 개의 템플릿(template rule)들로 구성되어 있고 각각의 템플릿들은 패턴정보를 가지고 있다. 여기서 패턴이란 XML문서내의 경로정보를 나타내며 XPath로 표현된다. 템플릿은 해당 패턴을 만났을 때 적용할 내용들을 기술하고 있으며 본 논문의 경우에는 생성할 HTML문서의 내용을 담고 있다.

XPath는 경로식(path expression)의 형태를 가지며 XML문서에서 특정 엘리먼트를 찾아주는 역할을 한다. 경로식은 루트로부터의 절대경로와 현재 위치로부터의 상대 경로 두 가 지 방식으로 표현 가능하다. 이때 경로의 표현은 엘리먼트의 이름으로 나타나며 부모와 자식관계는 '/'를 이용하여 나나태고 조상과 후손관계는 '//'를 이용하여 나타낸다. "/newspaper/sports//MLB"와 같은 XPath가 있다면 이 식은 루트 엘리먼트로부터 절대경로로 표현된 것이고 newspaper의 자식 엘리먼트로 sports가 존재하며 sports의 후손 엘리먼트로 MLB가 존재함을 나타낸다. "sports/baseball"과 같은 XPath가 있다면 이는 상대경로에 의한 표현이며 현재 경로를 기준으로 자식경로에 존재하는 sports를 찾고 다시 그 자식 엘리먼트인 baseball을 찾아나가게 된다.

그림 2에 나타난 XSLT 템플릿의 예를 그림 1의 XML에 적용해 보면 이 템플릿은 패턴 정보로 "sports"라는 XPath를 가지고 있기 때문에 sports 엘리먼트에

```
<xsl:template match ="sports">
  <h1> Sports Section</h1>
    <xsl:apply-templates select="baseball"/>
</xsl:template>
<xsl:template match ="baseball">
  <h3> MLB</h3>
    <xsl:value-of />
</xsl:template>
```

그림 2 XSLT 템플릿의 예

해당 규칙을 적용한다. 규칙의 내용을 보면 HTML 태그를 생성하고 재귀적으로 다시 템플릿을 호출한다. 이때 호출하는 템플릿은 패턴정보로 "baseball"을 가지고 있는 템플릿이 되고 상대경로가 적용되기 때문에 현재 템플릿을 적용한 엘리먼트의 자식 엘리먼트인 baseball 엘리먼트에 해당 규칙을 적용하게 된다. 이와 같은 방법으로 XML문서 내에서 일부만을 선택하여 HTML의 형태로 보여주는 것이 가능하다.

본 논문에서 XPath는 사용자 정보를 나타내는데 사용되어 문서내에 해당 부분에 대한 선호도를 표시할 수 있게 한다. XSLT는 이러한 사용자 정보를 바탕으로 문서내의 일부분만을 추출하여 HTML의 형태로 사용자에게 제공한다.

3. 관련 연구

문서 필터링 분야에서 사용자의 정보(user profile)를 어떻게 모델링할 것 인지와 이를 기반으로 사용자에게 원하는 문서만을 제공하는 기법에 관한 많은 연구들이 있어왔다[8-10]. 텍스트로 이루어진 문서 내에서 사용자의 취향을 나타내는 방법으로는 사용자의 취향을 몇 개의 키워드의 조합으로 나타내는 방법이 사용되었다. 그리고 이 키워드들을 어떻게 처리하느냐에 따라 불리언 모델(boolean model)이나 벡터 스페이스 모델(vector

space model), 확률 모델 등이 사용되었다[11]. 이밖에 정보 검색 분야에서 키워드 뿐 아니라 장이나 절, 문단과 같은 문서의 구조를 이용하여 검색하는 연구로는 [12, 13] 등이 있었다. 하지만 XML이 가진 구조 정보의 경우 위의 연구들과 같은 형식적인 정보뿐 아니라 다른 정보를 표현할 수 있기 때문에 새로운 접근 방법이 필요하다.

최근 들어 XML이 여러 분야의 정보교환 표준으로 채택되면서 XML로 이루어진 문서의 교환이나 XML문서의 관리를 위한 많은 노력이 있어왔다. 아파치의 cocoon[14]은 데이타베이스 내의 데이타를 XML문서로 가공하고 여러 형태의 웹 브라우져가 읽을 수 있도록 서버상에서 변환해 주는 시스템이다. 또한 컨텐츠 신디케이팅 분야에서는 자신들이 보유한 정보를 XML로 가공하여 여러 고객사들에게 공급하고 있다. 대표적인 예로써 로이터통신의 IDS[15], AP통신의 XML솔루션[16] 등이 있으며 이들은 NewsML[17], SportsML[18]과 같은 DTD표준에 기반을 두고 문서를 전송하고 있다.

이러한 시스템들은 XML문서의 관리와 출판에 중점을 두고 있지만 사용자에게 원하는 정보만을 제공하는 필터링의 관점과는 거리가 있다. cocoon과 같은 출판 시스템들은 XML 문서를 여러 가지 포맷으로 바꾸어 제공하는 컴포넌트들로 이루어져 있지만 사용자 정보를 관리하지 않는다. 컨텐츠를 중계하는 신디케이터들이 사용하고 있는 시스템들은 고객사들에게 원하는 정보를 해당 분야의 표준 XML형태로 제공하고 있다. 그러나 이들이 제공하는 서비스는 고객이 신청한 상품을 패키지의 형태로 제공하는 것일 뿐 개별 XML문서 대한 필터링을 지원하지 않는다. 고객사가 선택한 장르로 분류되는 정보들을 표준 XML화 하여 전송하거나 고객이 접속하여 받아가도록 하고 있다.

사용자 정보를 관리하여 XML문서를 제공하기 위한 연구로서는 [19,20] 등이 있다. 이 두 연구에서는 사용자 정보가 XPath로 기술되어 있다고 가정하고 SDI(Selective Dissemination of Information)환경에서 사용자에게 제공할 문서가 사용자에게 적합한지 판별하기 위한 사용자 정보 색인 구조를 제안하였다. 위 연구들은 사용자 정보로 나타난 XPath가 XML문서내에 존재하면 해당 XML문서 전체를 사용자에게 제공한다고 가정하였다. 정보 검색 분야에서 텍스트로 이루어진 문서의 일부를 검색 결과로 제시하는 연구가 있었지만[21] 이는 질의를 포함한 문헌에 대한 요약이고 어떤 구조적 정보에 의한 문서의 분할이 아니다.

본 논문과 위 연구들과의 차이점은 다음과 같다. 첫째, XML문서를 필터링할 때 사용자 정보를 모델링하는 방법을 제시하였다. [19, 20] 역시 사용자 정보를 XPath로 표현하였지만 임의의 XPath로 표현된 사용자 정보는 의미는 같아도 구조가 다르거나 엘리먼트의 이름이 다른 문서에 적용하기 곤란하다. DTD를 따르는 문서의 경우 DTD를 이용해 사용자 정보를 문서내의 존재 가능한 경로로 표현하여 이러한 사용자 정보 표현의 문제를 해결하였다. 둘째, 여러 문서 중에서 사용자가 원하는 부분만을 찾아내어 이들을 재구성하였다. 본 논문에서는 XML의 구조 정보를 이용하여 장이나 절과 같은 구조가 아니더라도 사용자가 원하는 각 문서의 일부분만을 추출, 통합 제공하고 있다. 이를 위해 XML의 질의어들중 하나인 XSLT를 이용하여 프레젠테이션 계층에서 개인화하기 위한 방법을 제시하였다. 기존의 웹 출판 시스템들은 XML로 이루어진 문서를 여러 형태로 출판하기 위한 다양한 기능들을 제공한다. 이러한 변환 과정에서 사용자 정보가 반영되어 문서의 일부만이 제공되도록 하였다.

4. 시스템의 구현

본 논문에서 구현한 시스템은 자바를 기반으로 웹 환경에서 구현되었으며 웹 서버측에서 동작한다. 뉴욕타임스[22]의 온라인 기사를 XML로 변환하고 기사별로 XSLT를 디자인하여 테스트하고 예제로 사용하였다. 각각의 기사는 복수의 DTD를 가진 경우를 테스트하기 위해 3개의 DTD를 따르는 것으로 재구성되었다.

시스템 관리자는 웹을 통해 시스템에 접속하고 XML로 이루어진 신문과 DTD 그리고 디자인에 해당하는 XSLT를 전송한다. 시스템은 각 DTD를 분석하여 시스템 관리자에게 DTD별로 문서의 분할 단위를 선택할 수 있는 폼을 제공한다. 시스템 관리자가 분할 단위를 선택하면 사용자가 사용자 정보를 입력할 수 있는 폼이 자동 생성된다. 정보를 제공받고 싶은 사용자는 웹을 이용하여 시스템에 접속한 후 시스템이 생성한 폼을 통해 자신의 정보를 입력한다. 이때 신문별로 원하는 부분을 선택하게 된다. 그러면 시스템은 이메일을 통해 사용자가 원하는 내용만을 추출 HTML로 변환하여 제공한다.

그림 3을 보면 시스템은 크게 두 부분으로 나뉘는데 첫째 부분은 DTD를 분석하여 XML문서를 어떤 단위로 쪼개어 제공할지를 문서 제공자가 선택하게 해주며 그에 따른 사용자 정보를 입력받기 위한 사용자 정보 입력 폼(user profile form)들을 생성해 낸다. 둘째 부분은 사용자가 입력한 사용자 정보를 바탕으로 기존의 XSLT를 재구성하여 사용자마다 다른 XSLT를 생성해 내는 부분이다. 이를 이용해 사용자별로 다른 HTML문서를 생성해내게 된다. 이 두 부분에 대해 다음 두 절에서 자세하게 설명한다.

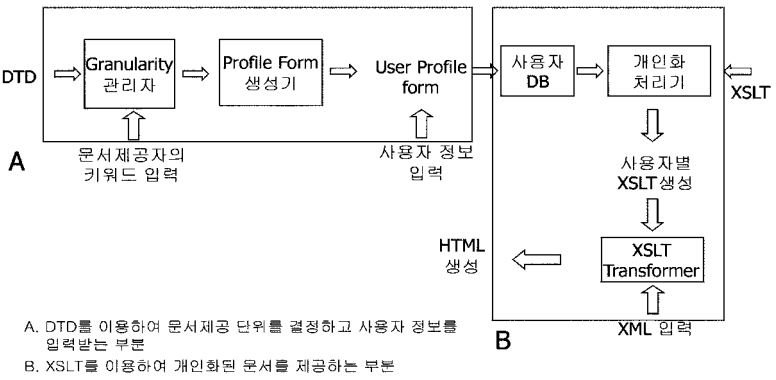


그림 3 시스템의 구조와 작업흐름

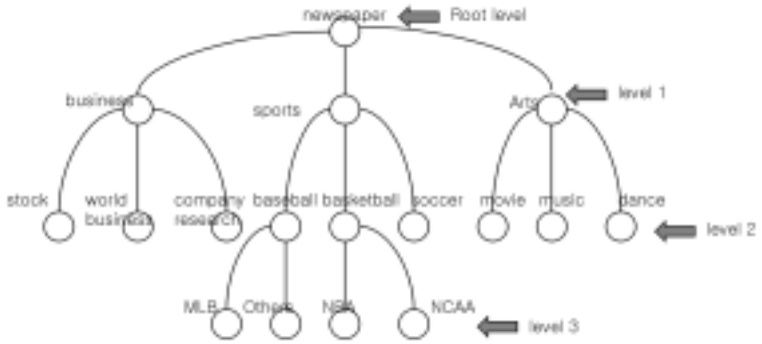


그림 4 XML문서의 제공 단위 분할

4.1 Granularity 관리자

Granularity 관리자에서 granularity는 XML문서를 트리구조로 보았을 때 어느 정도 레벨까지 XML문서를 분할할 것인가에 의해 결정된다.

그림 4를 보면 신문을 기술한 XML문서의 예가 나와 있는데 문서제공자는 문서를 받아볼 사용자가 원하는 부분을 어떤 단위로 선택을 하도록 할지 결정한다. 즉 사용자가 'business', 'sports', 'arts' 세 가지 섹션 중에 선호도를 선택하게 하려면 문서 제공자는 level 1까지의 경로정보를 사용자의 인터페이스 부분이 될 폼(form)문에 집어넣어야 한다. 더 세부적인 사항을 사용자에게 선택할 수 있도록 하려면 level 2 또는 level 3까지의 경로정보를 기술하여야 한다. 이때 일괄적으로 같은 레벨을 적용하는 것이 아니라 문서제공자가 그림 4에서 스포츠섹션은 level 3까지 세부적으로 선택 가능하도록 하고 나머지 business나 arts 섹션의 경우에는 섹션단위 즉 level 1에서 선호도를 선택하게 할 수도 있다.

본 논문에서 제공하는 XML문서에는 DTD정보가 있다고 가정하였으므로 DTD로부터 가능한 문서내의 경로정보를 모두 추출해낸 뒤, 문서제공자는 자신이 원하는 granularity를 결정하고 그에 따른 경로에 대해 각각 키워드를 부여한다.

예를 들어 문서제공자가 전체 문서의 granularity를 일괄적으로 level 2로 결정했다고 가정해 보자. 그러면 문서제공자는 level 1과 level 2에 해당하는 모든 경로에 대해 각각 일반 사용자가 쉽게 이해할 수 있는 키워드를 부여한다. "/newspaper/sports/baseball"이라는 경로에 대해 National League라는 키워드를 부여했다면 일반 사용자는 키워드를 선택하는 폼에서 문서내의 구조정보를 알 필요 없이 National League라는 키워드를 선택하기만 하면 된다. 사용자의 선택은 해당 경로 밑에 존재하는 모든 엘리먼트들을 제공받는 문서에서 보여주는 역할을 한다. 이때 경로에 부여하는 키워드는 간단한 설명이 될 수도 있고 문서의 내용을 대표할만한 몇 개의 키워드가 될 수도 있다. 사용자의 선택은 해당 경로의 후손으로 존재하는 모든 엘리먼트들을 제공받는 문

```
Input: DTD DTD파일
Output: ht Element들이 저장된 HashTable, root 루트 엘리먼트
ht = 각 element를 key로 하고 해당 element의 subelement들을 value로 가지는 해쉬테이블
list = DTD내의 모든 element를 저장할 리스트
e = DTD내에 나타나는 엘리먼트
s = e의 subelement
Procedure MakeDTDGraph(DTD)
while not end of DTD do
    e_i = ReadNextElement(DTD)
    ht. key_i ← e_i
    ht. value_i ← e_i.s
    list.add( e_i)
end while
FindRootElement(ht, list)
{루트에 해당하는 element를 찾아준다.}
PreorderTraversal(root)
{루트엘리먼트부터 전위순회를 하여 문서의 계층적인 구조를 보여준다.}
end procedure

Procedure FindRootElement(ht, list){
while not end of list do
    e_i = list.get(i)
    e_i.s = ht.Lookup( e_i )
    mark elements in list which are same as e_i.s
    (subelements of e_i )
end while
root = find a element in list which is not marked
end procedure
```

알고리즘 1 DTD를 parsing하여 문서의 구조를 계층적 형태로 보여주는 알고리즘

서에서 보여주는 역할을 한다.

이와 같은 작업이 가능하도록 하기 위해 Granularity 관리자는 DTD를 읽어들여 DTD를 그래프 구조(Directed Acyclic Graph)로 만들고 시스템관리자가 선택 가능한 문서내의 경로정보를 계층적 형태로 보여주며 이를 위해 알고리즘 1을 적용한다.

알고리즘 1은 [23]을 이용하여 구현되었으며 알고리즘 1을 적용하기 위해 필요한 가정은 다음과 같다.

가정 1 DTD에서 엘리먼트간에 싸이클(cycle)이 형성되는 경우를 배제한다.

가정 2 DTD에 나타나는 정규식(regular expression)을 단순화 한다.

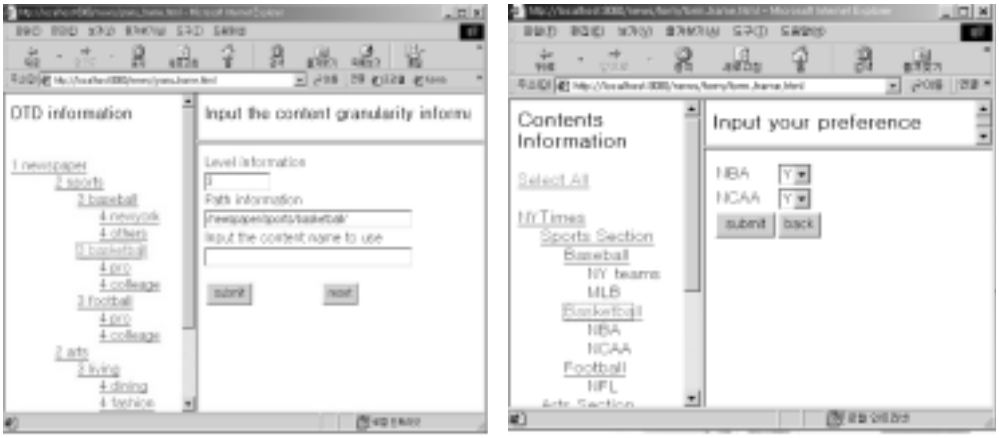
가정 1에서 싸이클이란 엘리먼트 A와 엘리먼트 B간에 부모자식 관계 또는 조상 후손관계가 동시에 성립되는 경우를 말한다. 이때 가능한 경로가 "./A/../B/../A/../B/.." 와 같이 무한히 반복될 수 있다. DTD만으로는 실제 경로가 어떻게 될 지 알 수 없지만 XML문서에서는 일정하게 반복되다가 끝날 것이다. 따라서 이런 경우를 배제하기 위해 해당 경로의 깊이가 30이상이 되지 않도록 알고리즘 1의 PreorderTraversal을 구현하여 가정 1이 성립되도록 하였다. 이러한 깊이의 제약은 전체 가능한 경로를 다 보여주지 못할 수도 있지만 granularity manager의 역할이 적당한 단위로 문서를 나눌 수 있는데 도움을 주는 것인 만큼 경로의 깊이에 제약을 두었다.

가정 2의 경우 DTD에 나타나는 정규식의 종류는 다음과 같으며 이를 규칙에 따라 단순화 한다.

(규칙1) r1, r2 => r1, r2
(규칙2) r1 | r2 => r1, r2
(규칙3) r+ => r
(규칙4) r* => r
(규칙5) r? => r

규칙 1에 나타나는 정규식은 변형하지 않고 그대로 적용한다. 규칙 2의 정규식은 여러 경로중 하나만이 생성될 것임을 나타나는데 이를 변형하여 생성 가능한 모든 경로를 추출한다. 즉 모든 경로에 시스템관리자는 일반 사용자가 선택할 수 있는 키워드를 부여할 수가 있다. 이때 제공되는 XML문서에 키워드를 부여한 경로가 나타나지 않는 경우가 있을 수 있고 사용자 또한 존재하지 않는 경로에 대한 키워드를 선택할 수 있다. 하지만 이런 경우 사용자 정보(XPath)를 이용하여 HTML을 생성할 때 XML문서 내에 존재하지 않는 사용자 정보는 무시되므로 문제가 발생하지 않는다.

규칙 3, 4, 5의 경우 엘리먼트가 존재하지 않거나 같은 이름의 엘리먼트가 여러 개 존재할 수 있음을 나타내는 데 모두 하나만 존재하는 것으로 변환한다. DTD가 적용된 XML문서에 해당 엘리먼트가 없는 경우는 규칙 2와 마찬가지이고 여러 개가 있는 경우는 모든 엘리먼트를 사용자에게 보여주거나 문서제공자가 애트리뷰트나 엘리먼트의 값을 이용하여 각각의 엘리먼트에 서로 다른 경로정보(XPath)와 키워드를 부여할 수 있다. 알고리즘 1은 스텝 10에서 스텝 15까지 DTD를 읽어들여 해쉬테이블에 엘리먼트와 그 엘리먼트가 가지고 있는 자식 엘리먼트들을 저장한다. 그리고 스텝 16에서 FindRootElement를 이용하여 루트엘리먼트를 찾는다. FindRootElement의 내용은 스텝 20에서 스텝 27까지 기술되어 있다. 루트에 해당하는 엘리먼트는 어떤 엘리먼트의 서브엘리먼트도 아니라는 사실을 이용한다. 스텝 22에서 리스트에 저장되어 있는 엘리먼트를 하나씩 꺼내고 스텝 23에서 스텝 24 사이에서 이를 이용해 어떤 엘리먼트의 서브엘리먼트에 해당하는 엘리먼트에 표시를 해준다. 이 과정이 끝나고 나면 *list*에는 루트 엘리먼



(a) Granularity 관리자 (b) 사용자정보 입력창

그림 5 XML문서의 단위 분할과 사용자 정보의 표현

트만이 표시가 안된 상태로 남아 있게 된다. 이렇게 얻은 루트 엘리먼트와 해쉬테이블을 이용해 스텝 17에서 그래프 구조(Directed Acylic Graph)를 전위순회하면 그림 5(a)의 왼쪽에서 문서의 전체적인 구조를 계층적 형태로 표시해 줄 수 있다.

그림 5(a)를 보면 좌측에 DTD를 이용하여 문서의 전체적인 구조를 보여주고 오른쪽에서 각각의 경로에 대해 사용자가 자신의 취향을 입력할 폼에 나타날 키워드를 정해주게 되어 있다. 그림 5에서 basketball이 여러 번 나타날 수 있는 엘리먼트이고 그 중 name이라는 애트리뷰트 값이 NBA인 경우를 구분해야 할 필요가 있다고 가정해 보자. 그렇다면 "/newspaper/sports/basketball[@name='NBA']"과 같이 XPath 부분에 애트리뷰트와 관련된 "[@name='NBA']"부분을 추가한 후 새로운 키워드를 부여하여 해결할 수 있다.

시스템 관리자에 의해 결정된 granularity, 각각의 문서내 경로(XPath로 표현)와 그에 상응하는 키워드들은 데이타베이스 내에 (level, path, keyword)와 같은 형태로 저장한다. 그림 3의 Profile Form 생성기는 이 정보를 이용하여 사용자가 취향을 입력할 폼들(user profile form)을 생성해낸다. 그림 5(b)는 실제로 생성된 폼을 나타낸다. 생성된 폼은 각각의 키워드에 대한 문서내의 경로정보(XPath)를 포함하고 있어 사용자가 이를 선택하면 사용자 데이타베이스에 XPath의 형태로 저장된다. 그림 5(b)의 좌측에는 시스템 관리자가 선택한 키워드들로 문서의 구조가 표시되며 문서의 전체 구조중 문서 제공자가 키워드를 부여한 레벨까지만 표시가 된다. 오른쪽에는 왼쪽 화면의 링크에 해당하는 사용자 정보 입력 폼이 표시된다. 사용자는 이러한 인터페이스를 통해 문서 내에서 자신이 원하는 특정 정보의 유무 또는 순서를 키워드만으로 결정할 수 있다.

4.2 개인화 처리기

개인화 처리기는 사용자정보를 바탕으로 XML문서를 HTML로 변환할 XSLT내의 패턴정보들을 재구성해준다. 이를 지원하기 위해서는 XSLT를 분석하고 템플릿 단위로 재구성해줄 효율적인 알고리즘이 필요하다. XSLT에서 지원하는 여러 가지 명세(specifications) 중 사용자가 필요한 부분을 추출해 내는 부분에 관해서만 언급한다. XSLT를 분석하기 위해 XSLT에 사용된 태그들을 아래와 같이 정의한다.

정의 1. (XSLT 태그의 분류) XSLT에 사용된 태그들은 다음과 같이 분류된다.

1. **Declaration Tag**
   <xsl:stylesheet>처럼 실제 HTML의 생성에 영향을 끼치지 않는 태그
2. **Template Tag**
   <xsl:template> 템플릿단위의 시작과 끝을 알리는 태그
3. **Instruction Tag**
   <xsl:apply-templates> <xsl:value-of>등과 같이 템플릿 내에서 실행해야 할 일들을 기술하는 태그들
4. **Design Tag**
   <html><h1>처럼 사용자에게 html문서로 보여질 태그들

위의 정의에 따라 XSLT내에서 사용되는 템플릿을 구성하기 위한 태그별 조합을 분류해보면 다음과 같다.

정의 2. (템플릿의 분류) XSLT 템플릿은 구성요소에 따라 다음과 같이 분류할 수 있다.

```
1. Declaration Tag Group
     템플과 상관없는 태그들의 집합
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
2. Instruction Tag Group
  템플릿 내에 명령어를 나타내는 태그만 있는 경우
<xsl:template match="sports">
    <xsl:apply-templates/>
</xsl:template>
3. Design Tag Group
템플릿 내에 디자인을 나타내는 태그만 있는 경우
 <xsl:template match="media-reference">
   <img src="{@data-location}" align="middle"/>
 </xsl:template>
4. Instruction Tag + Design Tag Group
템플릿 내에 명령어와 디자인을 나타내는 태그가 있는 경우
<xsl:template match="sports">
    <h1><i>SportsSection:baseball</i></h1>
           <xsl:apply-templates select="baseball"/>
         </xsl:template>
```

위에서 분류한 템플릿의 태그별 정의와 태그 조합의 분류에 따라 하나의 템플릿 단위를 나타내는 자료구조 t_node를 정의하였다. 개인화 처리기는 SAX API[24]를 이용하여 구현되었으며 SAX parser[25]는 XSLT의 태그들을 차례로 읽어 들여 앞서 분류한 태그들 중 어디에 속하는지를 판별하고 새로운 템플릿 단위를 생성하거나 생성되어 있는 템플릿의 내부 규칙에 해당하는 정보들을 채워 나간다. SAX API는 이벤트방식(event-driven)으로 이루어지기 때문에 태그가 시작될 때와 태그가 닫힐 때 불리는 startElement()와 endElement()를 t_node를 생성할 수 있도록 구현하였다.

그림 6의 왼쪽 XSLT는 알고리즘 2에 의해 그림 6의 오른쪽과 같은 자료구조로 변환된다. t_node의 pattern 값은 사용자 정보와 비교되어 t_node의 내용을 재구성하는데 사용된다.

전체 XSLT를 분석하고 나면 위와 같은 t_node들이 list에 저장된다. 사용자 정보에 의해 기존의 XSLT에서 새롭게 바뀌는 부분은 i_node부분이다. XSLT에서 템플릿은 재귀적으로 호출되며 템플릿이 가지고 있는 패턴 즉 경로를 기준으로 하위 엘리먼트에 대한 명령을 수행한다. 따라서 XML문서에서 사용자가 원하는 부분만을 나타내거나 순서를 바꾸기 위해서는 사용자 정보에 기술된 경로의 부모 경로에 해당하는 패턴을 가진 템플릿을 찾아 이 템플릿의 내부 정보를 바꾸어 주어야 한다.

그림 7의 사용자 정보 테이블을 보면 사용자의 선택이 path에 저장되어 있고 order는 해당 path가 같은 레벨에서 어떤 순서로 나타나야 할지를 표시하고 있다. 0은 사용자가 순서없이 보기를 희망한 경우이고 -1은 제

startElement() endElement(): SAX API procedure
Input: *xslt* XSLT 파일
Ouput: *list* 템플릿 오브젝트들이 저장되어 있는 리스트

t = name of tag
t1 = declaration tag *t2* = template tag
t3 = instruction tag *t4* = design tag
t_node = an unit of template, which includes several *i_nodes*
i_node = an unit of instruction, which includes several design tags
pattern = each *t_node* has unique pattern(path) and *i_node* may have it or not
type = each *i_node* has instruction type
design = design tags which belong to instruction tag
buffer = buffer for design tags

Procedure startElement(*t*)
  if *t==t1* **then**
    add *t1* to the *list*
  else if *t==t2* **then**
   create *t_node* and set *pattern*
   add *t_node* to the *list*
  else if *t==t3* **then**
   get *t_node* from the *list*
   create *i_node* and set *type* and set *pattern* for *i_node*
   if there is design tag(s) in *buffer*, set *design* for *i_node* and clear *buffer*
   add *i_node* to *t_node and* add *t_node* to *list*
  else
    if *buffer* is not empty and preceding tag is closed *t4* **then**
      get *t_node* from the *list* and get last *i_node* from *t_node*
      set *design* for *i_node* and clear *buffer*
      add *i_node* to *t_node* and add *t_node* to *list*
    else
      add *t4* to *buffer*
    end if
  end if
end procedure
Procedure endElement(*t*)
  if *t==t1* or *t==t3* **then**
    do nothing
  else if *t==t2* **then**
    if *buffer* is not empty **then**
      get *t_node* from the *list* and get last *i_node* from *t_node*
      set *design* for *i_node* and clear *buffer*
      add *i_node* to *t_node* and add *t_node* to *list*
    end if
  else
    add *t4* to *buffer*
  end if
end procedure

알고리즘 2 XSLT를 파싱하여 템플릿 오브젝트의 리스트로 바꾸어주는 알고리즘

```
<xsl:template match="newspaper">
  <H1>Sports Section</H1>
      <xsl:apply-templates select="sports"/>
  <H1>Arts Section</H1>
      <xsl:value-of select="arts"/>
 </xsl:template>
```

그림 6 알고리즘 2에 의한 XSLT의 t_node 오브젝트로의 변환

id	level	path	order	dtd
swyoo	3	/news/sports/baseball	1	1
swyoo	2	/news/sports	0	1
swyoo	3	/news/sports/basketball	2	1
swyoo	2	/news/arts	-1	2

그림 7 사용자 정보 테이블의 예

공받는 문서에 나타나지 않기를 희망한 경우이다. dtd는 사용자의 선택한 경로가 어떤 DTD로부터 생성된 것인가를 나타내고 있다. 이를 바탕으로 알고리즘 2에서 정의한 t_node를 재구성하는 알고리즘이 알고리즘 3에 기술되어 있다.

알고리즘 3은 사용자별 프로파일에 적용되어 DTD별로 새로 구성된 XSLT를 만들어 내게 된다. 먼저 리스트에 있는 t_node를 하나씩 꺼낸 후 t_node가 가진 pattern값을 취한다. 스텝 13에서는 pattern의 level값을 검사하여 이 값이 사용자 정보 테이블에 기술된 최대 레벨 값보다 작은지 검사한다. 그 이유는 사용자 정보에 나타난 path값의 부모 경로에 해당하는 t_node를 재구성하여야 하므로 사용자 정보보다 더 큰 level값을 가진 t_node들은 검사할 필요가 없다. 이때 t_node의 pattern값은 상대경로일 수도 있고 절대 경로일 수도 있으므로 상대경로인 경우는 DTD에서 얻어냈던 경로 정보들과의 비교를 통해 pattern의 레벨 값을 계산한다. 스텝 14에서 사용자 정보로부터 t_node의 자식 경로에 해당하며 path가 t_node의 pattern을 포함하고 있는 레코드들을 찾는다. 스텝 16에서 레코드의 order값을 꺼내고 order가 0이면 t_node가 가진 i_node의 pattern중 해당되는 pattern이 있는지 검사하여 없을 경우 이를 생성해준다(스텝17-18). 사용자 정보가 -1이면 해당되는 pattern을 가진 i_node를 삭제해준다(스텝 19-20). 사용자가 순서를 명시해준 경우에는 해당 pattern을 가진 i_node의 순서

```
// Input: list  t_node들이 들어있는 리스트
// Output: list 재구성된 t_node들이 들어있는 리스트

max: max level from user profile table
rs: result set from user profile table
pattern: path expression   p_level: level of pattern
order: order from table
path: path from table
level: level from table
Procedure ReorganizeXSLT(list)
while not end of list do
  t_ node_i=list.get()
  pattern=t_node.getPattern()
  if p_level < max then
    get rs from table where level=p_level+1
    while not end of rs do
      get order from rs_i
      if order==0 then
        if there is no i_node which pattern is
          rs_i.path, create i_node
       else if order==-1 then
        delete i_node which pattern is rs_i.path
       else
        update i_node order information in t_node
       end if
      end while
    end if
  end while
end procedure
```

알고리즘 3 사용자 정보에 따른 XSLT의 재구성

를 바꾸어 주거나 없는 경우는 생성해준다(스텝 21-22). i_node를 새로 생성할 때 type정보는 디폴트값이 apply-templates이다. 사용자별로 t_node의 리스트를

재구성하고 이로부터 새로운 XSLT를 생성해낸다.

최종적으로 XML문서와 함께 사용자별로 다른 XSLT를 XSLT processor[26]가 처리하면 HTML이 생성된다. HTML은 신문에 따라 생성되며 이를 사용자에게 이메일로 전송하게 된다.

5. 결론 및 향후 연구

향후 XML문서로 표현된 정보가 증가하면서 XML로 이루어진 문서 중 관심 있는 문서만을 제공하는 서비스도 활성화될 것이다. 우리가 구현한 시스템 이때 사용자 정보를 어떻게 표현할지와 XML문서의 구조 정보를 어떻게 이용할지에 대한 방법을 제안한다. 첫째, 사용자 정보는 문서의 구조정보로 대응시킨다. 문서의 구조정보로 대응되는 사용자 정보는 해당 문서의 특정 부분을 가리킬 수 있도록 해준다. 둘째, 사용자 정보가 구조정보로 표현되므로 문서의 전체가 아닌 일부를 가져올 수 있다. 따라서 여러 개의 문서에서 해당되는 부분만을 추출하여 제공 가능하다. 셋째, 사용자가 본인의 사용자 정보를 입력할 때 구조 정보의 복잡성을 고려하여 키워드만을 선택하게 한다.

현재 본 논문은 XML의 구조정보에 초점을 맞추고 있다. XML문서에는 구조 정보뿐 아니라 텍스트 엘리먼트가 가지는 내용정보도 있다. 따라서 기존의 사용자 정보 표현 방식이나 필터링 방식을 내용정보에 이용하여 구조정보와 내용정보를 모두 이용하는 연구도 필요할 것이다.

참 고 문 헌

[1] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XML-QL: A Query Language for XML. http://www.w3.org/TR/NOTE-xml-ql, 1998.

[2] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. International Journal on Digital Library, 1(1), 4, 1997.

[3] J. Clark and S. DeRose. XML Path Language (XPath) v1.0. http://www.w3.org/TR/XPath

[4] J. Clark XSL Transformation v1.0. http://www.w3c.org/TR/xslt.

[5] Dan Suciu. Semistructured data and XML in FODO, 1998.

[6] S. Abiteboul, P. Buneman and D. Suciu, "Data on the Web," 38-44. Morgan Kaufmann Publishers. San Francisco, CA, 2000.

[7] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. Proc. ACM SIGMOD Conf., 1996.

[8] M. Franklin and S. Zdonik. Data in Your Face: Push Technology in Perspective. Proc. ACM SIGMOD Conf., 1998.

[9] P. W. Foltz and S. T. Dumais. Personalized Information Delivery: an analysis of information filtering methods. CACM, 35(12):51-60, December 1992.

[10] T. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. ACM TODS, 24(4): 529-565, 1999.

[11] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison Wesley. 1999.

[12] Burkowski F. Retrieval activities in a database consisting of heterogeneous collections of structured texts. In SIGIR 1992.

[13] Gonzalo Navarro and Ricardo A. Baeza-Yates. Proximal nodes: A model to query document database by content and structure, Information Systems, 15(4), 1997.

[14] xml.apache.org. Coccon. http://xml.apache.org/cocoon/index.html, 2002.

[15] Reuters. Reuter Internet Delivery System. http://about.reuters.com, 2002.

[16] Associated Press. AP MegaSports XML Solution http://www.apdigitalnews.com/megasports, 2002.

[17] Internaltional Press Telecommunications Council. News Industry Text format. http://www.itpc.org, 2001.

[18] International Press Telecommunications Council. XML DTD for sports, http://www.sportsml.com, 2001.

[19] M. Altinel and M. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," VLDB 2000: 53-64.

[20] Chee Yong Chan, Pascal Felber, Minos N. Garofalakis, and Rajeev Rastogi. Efficient filtering of XML documents with XPath experssions. in ICDE 2002.

[21] Julian Kupiec, Jan O. Pedersen and Francine Chen. A trainable document summarizer. In SIGIR, 1995.

[22] The New York Times, http://www.nytimes.com/, 2001.

[23] Mark Wutka. DTD parser. http://www.wutka.com/dtdparser.html, 2000.

[24] Megginson Technologies, "SAX 1.0 : a free API for event-based XML parsing," http://www.megginson.com/SAX/index.html, 1998.

[25] Xerces2. Java XML Parser, http://xml.apache.org/xerces2-j/index.html, 2001.

[26] Xalan, Java XSLT processor, http://xml.apache.org/xalan-j/index.html 2001.

유 상 원

2000년 서울대 컴퓨터공학과(학사). 2002년 서울대 전기·컴퓨터공학부(석사) 2002년~현재 서울대 전기·컴퓨터공학부 박사과정. 관심분야는 데이타베이스, XMLIR

이 형 동

1997년 홍익대 컴퓨터공학과(학사). 1999년 서울대 컴퓨터공학과(석사). 1999년~현재 서울대 컴퓨터공학부 박사과정. 관심분야는 데이타베이스, 정보검색

김 형 주

정보과학회논문지 : 컴퓨팅의 실제
제 9 권 제 3 호 참조