학습 추론을 이용한 GUI 기반의 HTML2XML 래퍼
(GUI-based HTML2XML Wrapperusing Inductive Reasoning)

장 문 성 † 정 재 목 † 최 일 환 † 김 형 주 ††
(Moonsung Zhang) (JaeMok Jeong) (Ilhwan Choi) (Hyoung-Joo Kim)

요 약 래퍼(wrapper)는 미리 입력된 추출 규칙을 바탕으로 특정 정보 소스에서 원하는 정보를 추출, 가공하는 모듈이다. HTML-XML 래퍼(HTML Wrapper for XML)는 HTML로 이루어진 웹 정보에서 특정 정보를 XML 문서 형태로 추출한다. 사람이 추출 규칙을 직접 작성하는 일은 단순 반복적이고 지루한 일이므로, 최소의 노력으로 쉽고 빠르게 이를 생성할 수 있어야 한다.

본 논문에서는 기존의 스크립팅 방식에 GUI를 통한 학습 추론 방법을 통합하여 추출 규칙 생성 작업을 최소화 하는 방법을 제시한다.

키워드 : Wrapper, XML, HTML, Induction, WWW, Web, Training

Abstract The 'wrapper' is a module that extracts and processes information from the specified data source by the pre-composed extraction rule. 'HTML Wrapper for XML' extracts information from the web source as the form of XML document. Since composing the extraction rule is a repetitious and tedious job, it should be done as easy and fast as possible.

This paper presents the method to minimize the composing job, which integrates GUI based training and scripting.

Key words : Wrapper, XML, HTML, Induction, WWW, Web, Training

1. 개 요

래퍼(wrapper)는 미리 입력된 추출 규칙(extraction rule)을 바탕으로 특정 정보 소스에서 원하는 정보를 추출, 가공하는 모듈이다.

우리가 필요로 하는 정보의 범위가 커지고 다양해 질수록 정보 소스의 종류와 형식도 다양해 진다. 예를 들어, 어떤 종류의 정보를 얻기 위해서 RDB, text 문서, 웹, e-mail 등을 모두 검색해야 할 때도 있다. 이렇게 다양한 정보 소스로부터 원하는 정보를 통합적으로 추출해 내기 위해서는 이들을 통일된 데이타 모델로 접근할 수 있는 방법이 필요하다. 이런 경우 각각의 데이타 소스에 그에 대응되는 래퍼를 통해 접근함으로써 원하는 바를 달성할 수 있다.

· 본 논문은 두뇌한국21 사업에 의해 지원받았음.
† 비 회 원 : 서울대학교 컴퓨터공학부
sw6ueyz@hitel.net
jmjeong@oopsla.snu.ac.kr
ihchoi@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2001년 6월 29일
심사완료 : 2002년 5월 28일

인터넷을 통한 웹 정보는 근래에 들어 가장 방대하고 범용적인 정보 소스로 여겨지고 있다. 웹 정보는 대부분 HTML[18] 문서로 제공되는데, 이 문서는 기계가 알아볼 수 있는 의미 위주의 형식이 아니라 사람이 웹 브라우저를 통해 시각적으로 인식하는 것을 목표로 하는 표현 위주의 형식으로 되어 있다. 따라서 질의 처리기와 같이 HTML 문서 내의 정보를 필요로 하는 모듈이 문서로부터 원하는 정보를 얻어서 처리하기가 곤란하다는 문제점이 있다.

최근에는 XML[17]이 정보를 표현하고 교환하기 위한 표준으로 자리잡고 있다. XML은 데이타를 입체적이고 유연하게 표현할 수 있으며 뛰어난 확장성과 간결함으로 인해 많은 연구가 이루어지고 있다. 거의 모든 데이타는 XML로 일관되게 표현될 수 있으며, XML로 표현된 데이타는 의미 위주의 표현 방식으로 인해 기계가 쉽게 분석하고 읽어 들일 수 있다.

HTML2XML 시스템은 HTML 문서로부터 정보를 추출하여 XML 형식으로 표현하는 래퍼[1]의 일종이다.

1) 편의상 이러한 래퍼를 HTML-XML 래퍼, 또는 HTML wrapper for XML 이라고 표기한다.

이러한 종류의 래퍼는 그 유용성으로 인해 많은 연구가 이루어 지고 있다. 그러나 그 대부분은 래퍼 자체의 성능이나 추출 규칙의 표현 능력의 범위에 집중한 나머지 추출 규칙을 쉽고 빠르게 생성하는 방법에 대한 연구는 다소 소홀히 여겨지고 있다. 본 논문에서는 복잡한 작업 없이 GUI를 통해 추출 규칙을 빠르게 생성하는 작업 모델을 제시하고 이것을 가능하게 하는 고유한 알고리즘을 설명한다.

본 논문은 2장에서 관련 연구에 대해 설명하며, 3장에서는 HTML2XML 시스템의 세부적인 설계 내용에 대해 설명한다. 4장에서는 본 논문의 시스템을 이용하여 래퍼를 생성하는 간단한 시나리오를 보이고, 마지막 5장에서 논문을 정리한다.

2. 관련 연구

래퍼와 관련된 이론들은 활발히 연구되고 있는 주제 중 하나다. 래퍼를 전체적인 시스템의 구성 요소로 보는 관점[4]에서는 래퍼 자체의 기법 보다는 이질적인 데이타가 통합되어 사용자에게 하나의 데이타로 보여지는 과정과 이러한 과정에서 필요한 각 모듈의 구성을 중점적으로 다루고 있다. 본 논문에서는 HTML-XML 래퍼를 위한 관련 연구에 한정하여 살펴본다. 최근 무선 인터넷 환경의 성장으로, HTML을 모바일 폰이나 PDA와 같은 무선 장치에서 사용하는 WML[16] 등으로 변환하는 연구[19]도 있으나, 기본 동작 원리는 다른 HTML-XML 래퍼와 동일하다.

HTML 문서에서 데이타를 추출할 때의 문제점은 앞에서도 언급했듯이 HTML 문서의 목적이 사람에게 시각적으로 정보를 보여주는 데 있다는 점이다. 원하는 정보를 추출하기 위해 래퍼는 포매팅 정보나 화면 구성 관련 내용, 이미지[2] 등의 정보를 제거해야 하며, 어느 부분이 실제로 얻고자 하는 정보인지를 판단해야 한다. 따라서 완전히 자동적으로 정보를 뽑아내는 래퍼를 설계하는 것은 사실상 불가능하며, 어느 정도 사람이 개입하는 것은 불가피하다.

기존 연구의 방향은 사람의 개입 정도에 따라서 다음과 같이 크게 세 가지의 흐름으로 나누어 볼 수 있다: (1) 사람이 거의 개입을 하지 않고 AI 적인 패턴 인식 기법을 사용하는 방법[1,2,3], (2) 몇 개의 샘플(sample) 작업을 통해서 래퍼를 학습시킨 후에 래퍼가 스스로 추출 규칙을 추론하도록 하는 방법[9,11,13], 그리고 (3) 사람이 직접 래퍼가 제공하는 문법에 맞는 언어(language)[3]를 통해 선언적, 또는 절차적으로 래퍼에게 추론 규칙을 제공하는 방법[5,7,8,12,14,15] 등이다. 후자로 갈수록 사람의 개입과 노력이 많이 필요하기 때문에 대부분의 경우[5,12,14,15] 작업을 돕는 GUI를 함께 제공하게 된다.

(1)의 경우에는 사람의 노력이 적게 들어간다는 장점이 있지만, 사람이 의도하는 바를 완벽하게 찾아내지는 못한다는 문제점이 있거나[1,2] 그 결과만으로는 구체적인 래퍼를 만드는데 직접적인 도움을 줄 수 있는 것이 아니다[1,3]. 추출에 실패한 경우 별다른 방법이 없기 때문에, (1)의 연구들은 주로 사람이 원하는 바를 정확하게 찾아내는데 중점을 두고 있다.

(3)의 경우에는 사실상 사람이 직접 래퍼에 지시를 하는 방식이기 때문에 원하는 바를 정확히 얻을 수 있고 정밀한 제어가 가능하다는 장점이 있지만, 그만큼 사람의 노력이 많이 필요하게 되는 것은 당연하다. 최근의 웹 페이지들은 점차 규모가 방대해 지고 있는데, 가령 한겨레 신문사(http://www.hani.co.kr)의 경우 메인 페이지에만 평균적으로 1700여 개의 태그가 사용되고 있다. 이와 같은 페이지에서 원하는 정보까지의 태그 경로를 추적하여 추출 규칙을 스크립트로 프로그래밍 하는 작업은 비록 GUI가 어느 정도까지 작업을 돕는다고는 해도 상당한 시간과 노력을 요하는 작업이 된다. 또한 일단 생성한 추출 규칙으로는 제대로 추출되지 않는 HTML 문서가 발견될 경우 복잡한 스크립트를 또다시 디버그 해야 한다. 따라서 이러한 방식의 연구들은 주로 강력하고 다양한 라이브러리와 편리하고 쉬운 구문을 제공하는데 중점을 두고 있다.

(2)의 접근 방식은 현실적으로 선택할 수 있는 대안이다. [9]에서 제안한 시스템은 이러한 방식으로 동작하는 래퍼의 모델을 사용하고 있다. [11]과 [13]에서는 HTML 문서 내의 특정 위치를 선택하는 작업을 학습에 의해 수행하도록 하였다. 이러한 방식은 (3)에 비해 추출 규칙을 작성하는 사람의 노력은 적게 들지만 아무래도 사람의 의도에 완전히 일치하는 추출 규칙을 학습시키기가 어려운 경우가 발생한다. 또한 정보의 위치를 찾아내는데 성공하더라도 이것을 다시 XML이나 다른 출력 포맷으로 가공하는 작업 자체까지 학습시키기는 어렵다는 점이 있다.

---

2) 정보가 포함된 이미지도 생각할 수 있지만 여기서는 정보가 아닌 이미지를 의미한다.

3) 언어와 스크립트(script)는 다른 개념이지만 여기서는 언어를 통해 추출 규칙을 입력하는 방식을 스크립트 작성에 의한 방식이라고 표현하도록 하겠다. 래퍼는 이 스크립트 내용을 그대로 수행한다.

본 논문에서는 (2)와 (3)의 접근 방법을 절충하여, 일반적으로는 GUI를 통해 추출 규칙을 학습시키고 이것이 불가능하거나 추가적인 정밀한 가공 작업을 원할 때는 스크립팅을 이용하여 나머지 추출 규칙을 직접 입력하도록 하는 방식을 사용하였다.

3. HTML2XML 설계

이 장에서는 HTML2XML 시스템의 전체적인 구성을 보이고, 가장 중요한 부분인 스크립트 엔진과 학습 추론 알고리즘에 대해 설명한다.

3.1 구성

앞서 말한 바와 같이, HTML2XML 시스템은 기존의 스크립팅 방식과 학습 추론에 의한 방법을 모두 사용하며 이들은 GUI를 통해 유기적으로 통합되어 있다. 사람이 GUI를 통해 작업한 동작(action)들은 일단 대응되는 JavaScript로 변환되어 순차적으로 저장되었다가 HTML 창이 생성되거나 래퍼가 가동될 때 차례로 수행된다. 학습 추론 알고리즘의 핵심인 '태그 선택 규칙(3.2.1절 참조)' 역시 스크립트 정보에 함께 저장되었다가, 그 스크립트가 수행되기 직전에 적절한 태그를 선택하여 스크립트의 인자로 넘겨 진다.

HTML2XML 시스템은 **[그림 1]**와 같은 구조로 이루어져 있다. HTML Fetcher와 Parser는 이름 그대로 HTML 문서를 다른 사이트나 로컬 파일로부터 가져와서 태그 트리 구조로 분해하는 모듈이다. Web Browser와 HTML Tree View는 HTML Parser에서 분해된 정보를 읽어서 화면에 각자의 포맷으로 표시한다. Action-Script Converter는 사람의 동작(action)을 적절한 스크립트로 번역하는 역할을 한다. Merger는 학습으로 얻어진 여러 개의 '태그 선택 규칙'들을 하나의 규칙으로 통합해 준다. Script List 는 Job List의 다른 이름으로, 모든 스크립트를 리스트로 보관하고 있는 모듈이며 이들 스크립트의 모음이 바로 최종 결과물인 '추출 규칙' 자체가 된다.

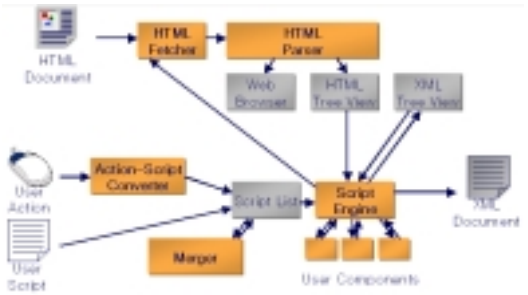


그림 1 HTML2XML 모듈 구성

3.2 학습 알고리즘

HTML 래퍼를 사용하여 정보를 추출하려고 하는 HTML 문서들은 유사한 구조의 서로 다른 HTML 페이지임을 가정하고 있다. 대표적인 것은 검색엔진에서 출력되는 결과 화면이나 신문사 사이트의 홈페이지다. 이들은 각각 검색어와 날짜에 따라서 그 내용은 달라지지만 화면 구성 자체는 항상 일정한 구조(structure)를 갖추고 있다. 이것은 래퍼가 정보를 추출할 때 중요하게 사용하는 단서이다. 항상 그 모습이 예측 불가능하게 변화하는 웹 페이지에서 일관되게 정보를 추출하는 것은 매우 어려운 작업이다.

따라서 본 HTML2XML 시스템에서 사용하는 학습 추론 알고리즘은 구조(structure)를 기반으로 하며 내용(content)을 기반으로 하지 않는다. 즉, 어떤 정보를 추출할 때 '이름', '전화번호'와 같은 문자열을 검색하려고 하는 것이 아니라 학습된 태그 구조의 패턴과 가장 유사한 태그 구조를 찾는다는 뜻이다. 내용을 검색하는 것은 해당 검색어가 문서의 다른 곳에 나타날 때 혼동이 일어날 수 있고, 'author'를 검색하기 위해서 'creator', 'writer'와 같이 검색어의 유의어에 대한 사전을 제공해야 한다는 문제점도 있다.

3.2.1 태그-아이템(tag-item), 인스턴스(instance) 그리고 규칙(rule)

알고리즘을 기술하기 전에 몇 가지 용어와 표기법에 대해 설명한다. 태그 트리 내의 어떤 태그에 대해서 (tag-name, tag-index, index)의 순서쌍을 가정할 수 있다. 이 순서쌍을 편의상 '태그 아이템(tag-item)'이라고 하자. 위에서 tag-name은 그 태그의 이름이고 tag-index는 그 태그의 부모 노드에서 같은 이름의 태그들 중 그 태그의 순서(index)이다. 또 index는 부모 노드의 모든 태그들 중 그 노드의 순서이다. 가령 **[그림 2]**에서 (b)의 <I> 태그는 (I,1,1)로 표기할 수 있고, (e)의 <LI> 태그는 (LI,2,3)으로 표기할 수 있다. 이제 루트 태그에서 어떤 태그까지의 경로(path)를 태그-아이템의 리스트로 표기할 수 있는데, 이를 인스턴스(instance)라고 하자. 사용자가 래퍼를 학습시키기 위해

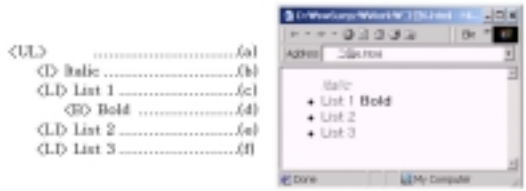


그림 2 태그 트리의 예

노드를 선택하면 하나의 인스턴스를 만들 수 있고 이것은 학습 과정에서 각각 하나의 학습 예제(training example)로 사용된다. 가령 **[그림 2]**에서 (d)에 대한 인스턴스는 (UL,1,1), (LI,1,2), (B,1,1)이다. tag-index와 index를 구분한 이유는, 가령 사용자가 (c)를 선택한 경우 '첫 번째 LI 태그'를 의도한 것인지 단순히 '두 번째 태그'를 의도한 것인지 불명확하기 때문에 두 가지 가능성을 모두 고려하기 위함이다.

'태그 선택 규칙(이하 '규칙'이라고 약칭한다. 추출을 위한 모든 작업의 집합인 '추출 규칙 extraction rule'과는 다른 것이다)'은 래퍼가 특정 태그를 선택하는 기준이 되는 자료구조로서 사용자가 제공하는 인스턴스를 통해서 계속적으로 갱신된다. 4장에 보일 예제의 후반부가 바로 이러한 갱신 과정의 하나이다. 규칙의 정확한 자료구조는 3.2.3절에서 예제와 함께 설명한다.

3.2.2 벌점 계산

태그 선택 규칙으로 선택되는 태그는 트리에 있는 모든 태그들 중에서 현재의 태그 선택 규칙과 가장 잘 들어맞는다고 생각되는 태그이다. 그 기준은 다음과 같은 불일치가 발생할 때마다 미리 정해진 벌점(penalty)을 주어 가장 벌점이 적은 태그를 찾는 것이다.

가령 태그 선택 규칙이 (H1,1,1),(B,2,3),(LI,1,1)이라고 할 때,

- **치환(replacement)**: 중간의 태그가 바뀜
  예: (H1,1,1),(FONT,2,3),(LI,1,1): B가 FONT로 치환되었다.
- **누락(omission)**: 중간의 태그가 빠져 있음
  예: (H1,1,1),(LI,1,1): B가 누락되었다.
- **추가(addition)**: 여분의 태그가 추가되었음
  예: (H1,1,1),(B,2,3),(UL,1,1),(LI,1,1): UL이 추가되었다.

의 세 종류의 불일치가 있다. 인스턴스의 각 태그-아이템에 대해서 가장 간단하게 벌점을 구하는 방법은 이들에 대해 모두 벌점 1점씩을 주는 방법일 것이다.

```
algorithm penalty_item( tag_set, tag_item, mode )
  return 1
```

이 경우 특정 규칙에 대한 특정 인스턴스의 벌점을 구하는 재귀적인 알고리즘은 다음과 같다. 이 때, 일단은 인스턴스와 규칙 모두 태그-아이템의 연결리스트로 구성된다고 가정하고, 위의 penalty_item의 tag_set 인자도 태그-아이템과 동일한 것이라고 가정하자.

이 재귀 알고리즘은 쉽게 dynamic programming 방식으로 바꿀 수 있으므로 알고리즘의 시간 복잡도는 rule의 길이가 m, instance의 길이가 n일때 O(mn)이다.

```
algorithm calc_penalty( rule, instance )
  if rule = null and instance = null then return 0
  if instance = null then addition_penalty <- infinity
  else addition_penalty <- penalty_item( rule, instance,
       "addition" ) + calc_penalty( rule, instance.next )
  if instance = null or rule = null then replace_penalty
       <- infinity else replace_penalty <- penalty_item
       ( rule, instance, "replacement" ) + calc_penalty
        ( rule.next, instance.next )
  if rule = null then omission_penalty <- infinity
  else omission_penalty <- penalty_item( rule, instance,
       "omission" ) + calc_penalty( rule.next, instance )
  return min( addition_penalty, replace_penalty,
       omission_penalty )
```

3.2.3 규칙의 통합과 학습 과정

3.2.2절 까지는 설명을 간단히 하기 위해 규칙('태그 선택 규칙'의 약칭)을 인스턴스와 동일한 자료 구조로 간주하였으나 지금부터는 HTML2XML 시스템에서 실제로 사용되는 규칙의 자료 구조를 설명하겠다. 사실 인스턴스는 규칙의 특수한 경우이다.

사용자가 다음과 같은 두 개의 인스턴스를 제시하였다고 하자. 래퍼는 두 인스턴스를 통합하여 하나의 규칙으로 만들어야 한다.

인스턴스#1: (H1,1,1),(B,2,3),(LI,1,1)

인스턴스#2: (H1,1,1),(FONT,2,3),(LI,1,1)

이제 규칙은 실제로 다음과 같은 모습을 가진다.

규칙: { (H1,1,1,2), (null,,,0) }, { (B,2,3,1), (FONT,2,3,1), (null,,,0) }, { (LI,1,1,2), (null,,,0) }

각 집합은 '태그-집합(tag-set)'이라고 부르며 규칙 자체는 이들 태그-집합들의 연결 리스트로 되어 있다. 태그-집합은 태그-아이템 순서쌍에 네 번째 차원 값인 가중치(weight)가 추가된 순서쌍들의 집합이다. 가중치는 그 태그-아이템이 태그-집합 내에서 어느 정도의 빈도로 선택이 되었는가를 말해주는 지표로, 한 태그-집합 내의 모든 가중치의 합은 동일하다. 위에서는 두 인스턴스를 각각 1 씩 동일한 가중치로 고려하여 모든 태그-집합의 가중치 합이 2가 되었다. null 태그는 그 태그-집합에서 태그를 선택하지 않고 지나갈 때 선택된다.

세 번째의 인스턴스를 위의 규칙에 추가하여 null 태그의 쓰임새를 확인할 수 있다.

인스턴스#3: (H1,1,1),(LI,1,1)

규칙: { (H1,1,1,3), (null,,,0) }, { (B,2,3,1), (FONT,2,3,1), (null,,,1) }, { (LI,1,1,3), (null,,,0) }

인스턴스#3을 중간에 null 태그가 있는 것처럼 가정하면((H1,1,1),(null,,),(LI,1,1)) 기존의 규칙과 가장 유사

하게 들어 맞으므로 이를 규칙에 추가하여 두 번째의 태그-집합의 null 태그의 가중치가 하나 증가하였다. 우리는 다음과 같은 네 번째의 인스턴스를 추가함으로써 치환, 누락, 추가의 세 가지 불일치에 대한 학습 과정을 모두 살펴보게 되었다.

인스턴스#4: (H1,1,1),(B,2,3),(UL,1,1),(LI,1,1)

규칙: { (H1,1,1,4), (null,,,0) }, { (B,2,3,2), (FONT, 2,3,1), (null,,,1) },{ (UL,1,1,1), (null,,,3) }, { (LI,1,1,4), (null,,,0) }

인스턴스#4에서 (UL,1,1)의 태그-아이템이 추가됨으로써 규칙에도 세 번째 위치에 새로운 태그-집합이 추가되었다. 인스턴스#1,#2,#3에서는 이 태그-집합에서 null을 선택한 셈이므로 null태그의 가중치는 즉시 3이 되고, 새롭게 선택된 UL 태그가 1의 가중치를 가지게 된다.

결국 학습이란 치환, 누락, 추가의 세 가지 불일치가 발생했을 때 이를 현재의 규칙에 추가(merge)하는 작업이다. 위의 작업을 알고리즘으로 정리하면 다음과 같다.

```
algorithm merge_helper( tag_set, tag_item, mode, all_
  weight )
  if mode = "addition" then
      ( tag_set이 가리키는 연결리스트의 헤더에 새로운
        태그-집합을 삽입하고, 이 태그-집합의 null
        태그에 all_weight를 weight값으로 준 다음
        tag_item을 추가 )
  else if mode = "omission" then
      ( tag_set의 null 태그의 weight를 1 증가 )
  else if mode = "replacement" then
      ( tag_set에 tag_item을 추가, 이미 존재하면 그
        weight만 1 증가시킴 )
  algorithm merge( rule, instance )
  cur <- (rule이 가리키는 태그-집합의 복사본)
  new_rule <- cur
  while ( rule != null or instance != null )
      ( calc_penalty 와 같은 방법으로 addition_penalty,
        replace_penalty, omission_penalty 를 구함 )
      if addition_penalty < replace_penalty
        and addition_penalty < omission_penalty then
        mode = "addition"
      else if omission_penalty < replace_penalty
        and omission_penalty <= addition_penalty then
        mode = "omission"
      else mode = :replacement"
      if mode = "addition" or "replacement" then
        instance = instance.next
      if mode = "replacement" or "omission" then rule
        = rule.next merge_helper ( cur,instance,mode,
        (rule의 각 태그-집합의 총 weight값) )
      if mode != "addition" then
        (cur의 다음 노드에 rule이 가리키는 태그-집합의
        복사본을 생성하고 그것을 cur이 가리키도록 함)
  return new_rule
```

가장 최근의 학습 내용을 가장 강하게 반영하기 위해 새로 더해지는 가중치를 현재 총 가중치에 비례하여 높은 값으로 책정할 수도 있을 것이다. 위의 merge함수에서 호출하는 calc_penalty 함수는 calc_penalty의 dynamic programming 알고리즘에서 미리 생성된 m * n 테이블을 사용하므로 시간 복잡도가 O(1)이다. 따라서 위의 merge 알고리즘은 시작 전에 한 번 calc_penalty를 호출한다고 가정하면 O(mn) + O(m+n)의 시간 복잡도를 가진다.

이제 지금까지 정의한 규칙을 사용하여 구체적인 벌점을 결정해야 한다. 처음에 무조건 1을 리턴하도록 가정했던 penalty_item의 실제 버전을 살펴보자.

```
algorithm penalty_item( tag_set, tag_item, mode )
  if mode = addition then return 1
  else if mode = omission then
      return 1 tag_set.null_tag_weight/tag_set.all_weight
  if tag_set.all_weight >= ANON_THRESHOLD then
      anon_penalty <- 1 (tag_set에서 tag_item.index와
      같은 index를 갖는 모든 태그-아이템의 weight의
      합)/tag_set.all_weight
  else anon_penalty <- 1
  if ( tag_set에 tag_item.name을 name으로 갖는
       태그-아이템이 없거나 있더라도 tag_index가 다름 )
       then return anon_penalty
  else return min( anon_penalty, 1
      ( 위의 if 문에서 찾은 태그-아이템의 weight값) /
        tag_set.all_weight )
```

추가(addition)의 경우에는 그 태그-아이템을 완전히 무시해야 하므로 1을 벌점으로 하고, 누락(omission)의 경우에는 null 태그를 선택하는 것이므로 1에서 null 태그의 가중치 비율만큼 벌점을 감한다. 치환(replacement)의 경우, 충분한 만큼(ANON_THRESHOLD: 실험적으로 3 정도의 값)의 가중치가 누적된 경우 태그 자체를 무명(anonymous)으로 보는 방법이 가능하다. 이것은 (B,2,3)과 (LI,1,3)을 동일한 세 번째 태그로 취급하는 것이다. 누락의 경우와 마찬가지로, 선택된 태그-아이템의 가중치 비율만큼 벌점을 감한다. 100%의 적중인 경우 벌점은 전혀 없다.

3.3 HTML2XML 스크립트

3.2의 알고리즘은 사용자의 GUI 작업을 통해 특정 노드를 선택하는 방법을 래퍼가 학습하는 원리를 기술한 것이다. 일단 이렇게 선택된 노드에 대해서, 시스템이 미리 정의해 둔 작업들 보다 정교한 작업 지시를 내리고 싶으면 사용자는 스크립트를 직접 입력하게 된다. 즉, 노드를 선택하는 단계 까지는 이러한 일을 하기에 적합한 학습 알고리즘에 의해 수행하고, 노드에 대한 처리는 스크립트를 사용하여 세밀하게 지시할 수 있다. 본 시스템

에서 내부적으로는 모든 작업이 그에 상응하는 스크립트로 기술되어 있으므로 추출 작업은 단순히 이들 스크립트를 순차적으로 실행하기만 하면 된다. 이번 절에서는 HTML2XML 시스템의 스크립트에 대해 설명한다.

HTML2XML 시스템에서 사용하는 스크립트는 대중적으로 널리 알려져 있는 JavaScript[6]이다. JavaScript는 HTML 문서에 동적인 인터페이스를 추가하기 위한 언어로서 많이 사용되며 대부분의 웹 브라우저에서 지원한다. 새롭게 문법을 정의하지 않고 JavaScript를 사용하는 이유는 크게 두 가지다. 현실적으로, 단순 반복적인 추출 규칙 생성 작업의 성격상 이 작업을 담당하는 사람은 고급 인력이 아닐 가능이 높다. 따라서 추출을 위한 새로운 언어를 습득하도록 하는 것 보다는 친숙한 언어를 사용하는 것이 더 효율적일 것이다. 두 번째 이유는 JavaScript 특유의 단순성과 확장성이다. 언어 자체에 다양한 라이브러리가 포함되기 보다는 사람이 정의한 객체들간의 상호 작용을 돕는 정도의 기능만 제공함으로써 대부분의 기능을 객체의 능력에 의존하고 있다. 바꾸어 말하면 사람이 얼마든지 필요한 객체를 추가함으로써 기능을 확장시킬 수 있다는 것이다. 실제로 HTML2XML 시스템에서는 사람이 자유롭게 객체를 추가하고 이를 스크립트에서 접근할 수 있다. 본 논문에서는 HTML2XML 시스템을 다양한 기본 제공(built-in) 객체들로 포화시키는데 노력하기 보다는 확장 가능성을 확보하는데 중점을 두기로 하였다. JavaScript는 기본적으로 String, Regular Expression, Math, Array, File 등의 객체를 내장하고 있으므로 이러한 객체까지 사람이 제공할 필요는 없다.

[표 1]은 HTML2XML 시스템의 스크립트에서 접근 가능한 객체와 각각의 메서드(method), 프로퍼티(property)를 일부 나열한 것이다. 트리를 계층적으로 방문하고 트리의 속성과 이름, 텍스트를 얻을 수 있는 가장 기본적인 메서드들과, 트리에 추가적인 태그를 추가하기 위한 Add- 로 시작하는 메서드들이 포함된다. 'htree'(HTML 태그 트리)에는 한 번에 여러 개의 HTML 문서를 AddUrl 메서드로 읽어 들일 수 있으므로[4], 일반적인 경우 htree.root 의 각 자식노드는 읽어 들인 HTML 파일들의 <BODY> 태그가 된다. 그 밖에, 시스템으로부터 명령행 인자를 읽거나 설정값을 읽어 들이기 위한 'system' 객체가 있다. 다음의 스크립트는 명령행 인자로부터 첫 번째 문자열을 읽어서 그 위

4) 'LoadUrl'이나 'GetUrl'이 아닌 'AddUrl'인 이유이다. 그러나 웹 브라우저 창에는 가장 마지막으로 읽어 들인 HTML 문서가 표시된다.

표 1 스크립트에서 접근 가능한 객체, 메서드, 프로퍼티

객체/클래스	메서드/프로퍼티	설명
'htree'(전역객체)	AddUrl(url)	루트 노드에 지정된 url을 로드하여 추가함
	root	루트 노드를 얻음
'xtree'(전역객체)	Save(filepath)	지정된 파일 패스로 XML 문서를 저장함
	root	루트 노드를 얻음
노드(클래스)	Add(name,value)	name 이라는 태그를 만들고 그 밑에 value 를 텍스트로 삽입함. name == null 이면 텍스트 노드만을 삽입하고, value == null 이면 태그만을 삽입한다. name 이 다른 노드라면 그 노드와 후손 노드들을 지정된 노드 밑에 삽입한다.
	name	노드의 태그 이름
	value	노드의 텍스트 내용
	firstChild	첫 번째 자식 노드
	lastChild	마지막 자식 노드
	nextSibling	다음 형제 노드
	prevSibling	이전 형제 노드
	parent	부모 노드
	firstAttribute	첫 번째 속성 노드 (노드 클래스를 반환함)
	lastAttribute	마지막 속성 노드 (노드 클래스를 반환함)
'system'(전역객체)	Debug(message)	스크립트 디버그를 위한 로깅 함수
	args(index)	index 번째의 명령행 인자를 얻음

치로 이동하고 <BODY> 태그 밑의 모든 태그의 태그명과 텍스트를 XML 태그 트리의 루트 노드에 추가하는 예제이다.

```
htree.AddUrl( system.args(0) );
var parent = htree.root.firstChild;    // get body tag
for ( child = parent.firstChild; child != null;
            child = child->nextSibling )
   xtree.root.Add( child.name, child.value );
```

실제로는 하나의 스크립트에 하나의 main 함수가 존재하고, 스크립트 엔진은 main 함수만을 수행한다. 사용자는 스크립트에 추가적인 함수를 정의하고 main 함수에서 그 함수를 호출할 수 있다. main 함수의 인자는 현재 작업중인 노드이다. 전역적인 사용자 정의 함수라면 인자가 없겠지만, 특정 노드에 대한 작업의 경우 그 노드를 인자로 넘겨 받는다. 3장에서 예제로 사용되었던 'Add this node to XML tree'와 같이 HTML 노드의 내용을 XML 노드로 복사하는 경우 양쪽 노드를 모두 인자로 넘겨 받게 된다.

여기서 중요한 점은 넘겨 받는 노드 자체는 스크립트에서 결정하는 것이 아니라 학습에 의해 만들어진 '노드

선택 규칙'을 통해서 결정된다는 것이다.[5] 정확한 노드를 선택하지 못하면, 스크립트를 수정할 필요가 없이 GUI를 통해서 재학습이 가능하다.

다음은 선택한 HTML 노드의 모든 후손 노드를 검색하여 H1~H6 의 태그 내용을 현재 선택된 XML 노드로 복사하는 사용자 정의 스크립트이다.

```
// global regular expression object, ignore case
var re = new RegExp( "H[1-6]", ig );
// recursive function
function FindAndAdd( hnode, xnode )
{
        if ( hnode.name.match(re) )
                xnode.Add( hnode.name, hnode.value );
        // recurse all children
        for ( child = hnode.firstChild; child != null;
                                child = child.nextChild )
                FindAndAdd ( child, xnode );
}
// main function
function main( hnode )
{
        xnode = xtree.root.Add( "search result" );
        FindAndAdd( hnode, xnode );
}
```

사용자 정의 객체를 사용하는 경우 복잡한 작업을 간결하게 만들 수 있다. 현재 HTML2XML 시스템에서 지적되는 약점은 하나의 텍스트를 하나의 노드로 취급한다는 점이다. 따라서 쉼표나 개행 문자(new line character)로 구분된 데이타는 직접 스크립트에서 문자열 처리 작업을 해야 한다는 불편함이 있다. 텍스트를 문장 부호로 분해하여 각각의 노드로 만들어 주는 사용자 정의 객체가 있다면 분해할 텍스트 노드에 대해서 다음과 같은 사용자 정의 스크립트를 수행하면 된다.

```
function main( node )
{ var myParser = ActiveXObject ( "MyParser.Sentence
                        Parser" ) ;
  var subtree = myParser.parse ( node.value );
  node.Add ( subtree, null ) ; }
```

4. HTML2XML 작업 시나리오

이 장에서는 HTML2XML 시스템을 이용하여 래퍼의 추출 규칙을 작성하는 몇 가지 시나리오를 생각하면서 작업 예를 살펴보도록 하겠다.

가령, 매일 한겨레 신문사 홈페이지(http://www.hani.co.kr)에서 헤드라인 기사의 제목을 추출해 내는 작업을 수행한다고 한다면, 다음과 같이 한다.

① **메뉴에서 file->new project를 선택**하여, 빈 생성 규칙을 생성한다. 가장 우측에는 현재 생성되고 있는 추출 규칙의 내용이 표시되며, 사용자가 하나의 동작(action)을 수행할 때 마다 그것에 대응되는 하나의 작업이 기록된다. 각각의 작업은 사용자의 동작을 적절한 JavaScript로 번역한 텍스트 데이타이다.

② **메뉴에서 file->new pane을 선택**하여, 샘플 HTML 파일을 읽어 들이기 위한 창을 생성한다. 창은 여러 개 생성할 수 있으며, 각각의 창에 서로 다른 샘플 HTML 파일을 읽어 들일 수 있다. 이 때 우측의 추출 규칙은 공유하므로, 사용자는 현재 생성하고 있는 추출 규칙이 여러 개의 HTML 파일에 적용될 때의 모습을 미리 확인할 수 있다.

③ **명령행[6] 내용을 묻는 대화 상자에 HTML 파일의 경로와 결과로 저장될 XML 파일명을 입력**한다. 각 HTML 창은 실제 래퍼가 추출 작업을 수행할 때 하나의 래퍼 프로세스에 해당된다. 따라서 각 창에 대한 명령행 인자가 다르기 때문에 따로 입력을 받아야 한다. [그림 3]에서 창의 가장 좌측은 지정한 HTML 파일을 보여주는 웹 브라우저이며, 중간 부분은 읽은 HTML 문서의 태그 트리[7], 그리고 가장 우측은 생성중인 XML 파일의 태그 트리가 표시된다.

④ **우측의 작업 리스트에서 팝업 메뉴로 AddUrl을 선택**한다. 이 작업은 실제로 HTML 문서를 읽어

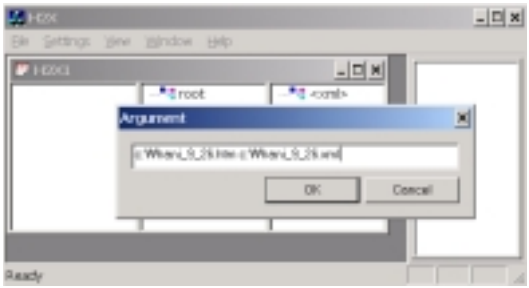


그림 4 새로운 URL을 명령행의 첫 번째 인자에서 읽어 들이는 작업

5) 전역적으로 동작하는 사용자 정의 스크립트의 경우 노드를 입력 받지 않으므로 스크립트에서 노드 선택작업까지 해야 한다.

6) 터미널 창에서 프로그램을 수행할 때 실행파일명 뒤에 추가적으로 입력하는 정보 : ls 1의 1 같은 것.

7) HTML 문서나 XML 문서는 태그가 계층 구조로 되어 있으므로 이를 트리 형태로 표시하면 구조를 쉽게 알아 볼 수 있다.

그림 4 새로운 URL을 명령행의 첫 번째 인자에서 읽어 들이는 작업

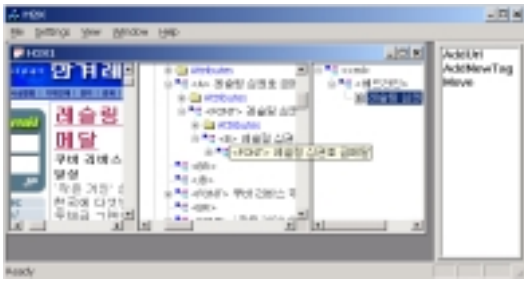


그림 5 헤드라인만을 추출하여 XML 트리를 구성함

들인다. 문서의 URL은 고정된 주소에서 읽을 수도 있고, 래퍼가 수행될 때 명령행에서 얻을 수도 있다. ③에서 명령행에 파일 경로를 입력했으므로 **[그림 4]**와 같이 명령행 인자 0 번(즉 첫번째 인자)에서 얻을 수 있다. 여기서 이 작업에 대한 내부 표현인 JavaScript 구문을 볼 수 있는데 이를 수정하여 기본 수행 작업을 덮어 씌울 수도 있다.

⑤ **웹 브라우저에서 헤드라인 기사를 마우스로 더블 클릭**한다. HTML 태그 트리에서 그 내용을 가지고 있는 태그가 자동으로 선택된다. 이것은 HTML2XML 시스템의 장점 중 하나로, 사용자가 웹 페이지의 태그 구조를 몰라도 웹 브라우저에서 보여지는 정보 그대로를 마우스로 클릭하여 선택할 수 있다.

⑥ **XML 태그 트리의 루트 노드에서 팝업 메뉴로 Add a new tag를 선택하고 '헤드라인' 태그를 만든다.**

⑦ **HTML 태그 트리의 선택된 노드에서 팝업 메뉴로 Add this node to XML tree를 선택**한다.

⑧ **작업 리스트의 팝업 메뉴로 Save XML tree를 선택하고 파일명으로 명령행 인자 1을 선택**한다. ③에서 두 번째 인자로 입력한 XML 파일명을 사용한다. 이 때 입력한 경로명을 가진 XML 파일이 생성된다.

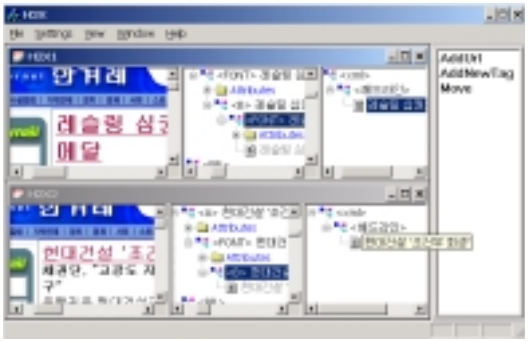


그림 6 두 HTML 문서에 대한 동시 작업

⑨ **메뉴에서 file->save를 선택해서 Test.h2x 라는 파일명으로 저장**한다. 이제 배치 파일이나 명령행에서 다음과 같은 실행문으로 지금까지의 작업을 반복할 수 있다.H2X.exe test.h2x <HTML 경로> <XML경로>

⑩ 테스트를 위해 새로운 창을 생성하여 다른 HTML 문서를 읽어 들여 볼 수 있다. 명령행에서 또 다른 HTML 파일 경로와 XML 파일 경로를 입력하면 창이 생성되는 순간 추출 규칙 내의 모든 작업 리스트가 순차적으로 수행되면서 **[그림 6]**처럼 해당 HTML 문서에서 추출한 내용이 XML 트리로 만들어지고 동시에 XML 파일로 저장됨을 확인할 수 있다.

마우스 작업에 익숙하다면 위의 과정은 모두 합쳐서 1분 내에 수행할 수 있다.

여기서 주목해야 할 점은 추출 규칙의 생성 작업이 쉽고 빠르다는 것 말고도 추출 규칙 적용 과정에 융통성이 있다는 것이다. **[그림 6]**에서 보듯이 추출되는 두 HTML 태그는 미묘한 차이가 있다. 즉, 하나는 -<FONT>-<B>-<FONT>-text 의 형태이고 다른 하나는 -<FONT>-<B>-text 의 형태를 가진다. 전자는 추출 규칙을 생성할 때 사용된 형태이고, 후자는 실제로 추출 규칙을 적용하면서 만나게 된 형태지만 양쪽 모두 추출 규칙에 가장 근접한 태그를 정확히 선택해 냈음을 알 수 있다. 이것은 3.2에서 설명한 '벌점 계산' 알고리즘에 의한 결과이지만 항상 100%의 결과를 보장해 주는 것은 아니다. 두 번째 시나리오는 이와 같이 정확한 태그를 찾지 못해 원래의 의도와는 다른 결과가 나왔을 때 학습에 의해 추출 규칙을 바로 잡는 과정을 보인다.

① **메뉴에서 file->open project 를 선택하고 해당**

추출 규칙 파일을 연다.

② **메뉴에서 file->new pane 을 선택하고 문제가 발생한 HTML 문서를 연다.** 위의 예제처럼 명령행 인자로 입력했을 수도 있고 고정 주소를 사용했을 수도 있을 것이다.

③ **우측의 작업 리스트에서 문제가 되는 작업을 선택하고, 웹 브라우저에서 올바른 정보를 더블 클릭한다.** HTML 태그 트리에서 직접 태그를 선택해도 된다.

④ **HTML 태그 트리에서 Merge를 선택하고 '확인' 버튼을 클릭한다.**

⑤ **메뉴에서 file->save로 저장한다.**

이제 새롭게 갱신된 추출 규칙은 문제가 되는 HTML 파일을 올바르게 처리할 뿐만 아니라, 기존의 추출 규칙과 새로운 추출 규칙의 공통점과 차이점을 효과적으로 분석하여 새롭게 입력되는 HTML 파일에 적용하기 시작한다.

5. 결 론

이 논문에서는 HTML 문서에서 원하는 정보를 추출하여 XML 문서를 생성하는 HTML2XML래퍼 시스템을 설명하였다. HTML2XML 시스템은 GUI를 통한 학습 추론 알고리즘을 사용하여 사람의 개입을 최소화 시키는 한편, 이것만으로 부족한 부분에 대해서는 직접 스크립트를 추가할 수도 있도록 하였다. GUI 작업은 직접 웹 브라우저와 XML 태그 트리를 마우스로 클릭해 가면서 작업하는 정도의 고수준이다. 또 추출에 문제가 발생하는 경우에는 잘못 동작하는 부분에 대해서만 GUI로 재학습시킬 수 있다. 스크립트 언어로는 대중적인 JavaScript를 사용하여 프로그래밍에 익숙하지 않은 사람도 쉽게 학습할 수 있고, 추가적인 사용자 정의 모듈을 쉽게 추가하고 사용할 수 있도록 하였다.

향후의 연구 방향으로는 XML DTD에 대한 처리가 남아 있다. 현재 HTML2XML 시스템은 XML DTD가 고려되고 있지 않다. 현재는 작업자가 원하는 XML DTD의 모양을 머리 속으로 상상하면서 XML 트리를 직접 구성해 나가야 한다. XML DTD를 입력 받아 XML 태그 트리의 골격을 자동으로 생성할 수 있다면 보다 정형화 되고 통일된 XML 뷰(view)를 중재자(mediator)에게 제공할 수 있을 것이다.

참 조 문 헌

[1] Brad Adelberg. NoDoSE A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents. In *SIGMOD Conference*, Pages 283-294, 1998.

[2] William W. Cohen. Recognizing Structure in Web Pages using Similarity Queries. In *AAAI/IAAI*, Pages 59-66. 1999.

[3] David W. Embley, Y.S.Jiang, and Yiu-Kai Ng. Record-Boundary Discovery in Web Documents. In *SIGMOD Conference*, Pages 467-478, 1999

[4] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database Techniques for the World Wide Web: A Survey. In *SIGMOD Record 27(3)*. Pages 59-74, 1998.

[5] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. Jedi: Extracting, and Synthesizing Information from the Web. In *CoopIS*, Pages 32-43, 1998

[6] James Jaworski. *Inside Secrets JavaScript & JScript*. 삼각형 프레스, 1999.

[7] JaeMok Jeong. Xws script language. Technical report, Seoul National University, Jan 2000. http://oopsla.snu.ac.kr/xweet/xws/xws-script.txt.

[8] Thomas Kistler, and Hannes Marais. WebL A Programming Language for the Web. In *WWW7/Computer Networks 30*, Pages 259-270, 1998.

[9] Craig A. Knoblock, Steven Minton, Jos Luis Ambite, Naveen Ashish, Pragnesh J. Modi, Ion Muslea, Andrew Philpot, and Sheila Tejada. Modeling Web Sources for Information Integration. In *AAAI/IAAI*, Pages 211-218, 1998.

[10] Nicholas Kushmerick. Regression testing for wrapper maintenance. In *AAAI/IAAI*, Pages 74-79, 1999.

[11] Nicholas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper Induction for Information Extraction. In *IJCAI*, Pages 729-737, 1997.

[12] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *ICDE*, Pages 611-621, 2000.

[13] Ion Muslea, Steve Minton, and Craig A. Knoblock. Active Learning Hierarchical Wrapper Induction. In *AAAI/IAAI*, Page 975, 1999.

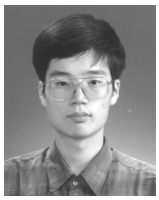
[14] Arnaud Sahuguet, and Fabien Azavant. Building Light-Weight Wrappers for Legacy Web Data-Sources Using W4F. In *VLDB*, pages 738-741, 1999.

[15] Arnaud Sahuguet, and Fabien Azavant. Web Ecology: Recycling HTML Pages as XML Documents Using W4F. In *WebDB*, pages 31-36, 1999.

[16] Wireless Application Protocol Forum Ltd, Wireless

Markup Language (WML) 2.0 Specification, 2001. http://www.wapforum.org/what/technical.htm
[17] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.
[18] World Wide Web Consortium (W3C). HTML 4.01 Specification, Dec. 1999. http://www.w3.org/TR/1999/REC-html401-19991224.
[19] 이승진, 김대건, 최린, 강철희, 확장성 있는 웹서비스를 위한 무선 응용 프로토콜 기반의 HTML Filter 구현, 한국정보과학회 논문집, Vol.28, No.1, pages 391-393, 2001.
[20] 전병선. *Microsoft Visual C++ 6.0 ATL COM Programming*. 삼양출판사 1999.

장 문 성
1999년 서울대학교 컴퓨터공학과 학사. 2001년 서울대학교 컴퓨터공학과 석사. 관심분야는 XML, 사용자 인터페이스



정 재 목
1994년 2월 서울대학교 자연과학대학 계산통계학과(학사). 1996년 2월 서울대학교 자연과학대학 전산과학과(석사). 1996년 ~ 현재 서울대학교 자연과학대학 전산과학과 재학중. 관심분야는 트랜잭션처리, 멀티미디어 DBMS, 클라이언트-서버 DBMS

최 일 환
1996년 서울대학교 컴퓨터공학과 학사. 1998년 서울대학교 컴퓨터공학과 석사. 1998년 ~ 현재 서울대학교 컴퓨터공학과 박사과정 재학중. 관심분야는 XML, 데이타베이스

김 형 주
정보과학회논문지 : 데이타베이스
제 29 권 제 1 호 참조