이화여자대학교 과학기술대학원
2005학년도
석사학위 청구 논문

CXquery를 이용한 스트리밍 XML 데이터 필터링

컴 퓨 터 학 과
김 소 라
2 0 0 6

CXquery를 이용한
스트리밍 XML 데이터 필터링

이 論文을 碩士學位 論文으로 提出 함

2006 年 1 月

梨花女子大學校 科學技術大學院

컴퓨터학과 金 沼 羅

김소라의 碩 士 學 位 論 文 을 認准 함

指導敎授 용 환 승 _____

審査委員 이 상 호 _____

용 환 승 _____

이 미 정 _____

梨花女子大學校 科學技術大學院

목 차

논문개요 ---------- v

Ⅰ. 서론 ---------- 1

1.1 연구 배경 및 목적 ---------- 1

1.2 연구 내용 ---------- 2

Ⅱ. 관련 기술 및 연구 동향 ---------- 4

2.1 XML 질의 언어 ---------- 4

2.2 스트리밍 XML 데이터 필터링 ---------- 5

Ⅲ. 필터링 시스템 설계 ---------- 8

3.1 질의 시스템 구조 ---------- 9

3.2 시스템 처리부 ---------- 11

Ⅳ. CYFilter시스템 구현 ---------- 23

4.1 시스템 구현 환경 ---------- 23

4.2 시스템 구성 ---------- 24

Ⅴ. 실험 및 분석 ---------- 36

5.1 실험 환경 및 데이터 ---------- 36

5.2 실험 평가 ---------- 37

Ⅵ. 결론 및 향후 과제 ---------- 42

참고문헌 ---------- 44

영문초록 ---------- 46

그 림 목 차

[그림 2.1] YFilter에서 질의의 NFA표현 ---------- 7

[그림 3.1] CYFilter 구조도 ---------- 10

[그림 3.2] 경로 생성 알고리즘 ---------- 12

[그림 3.3] 예제 XML문서의 DOM 트리 ---------- 13

[그림 3.4] [그림 3.3]의 PP파일 ---------- 13

[그림 3.5] 논리 연산자 and 사용시, 두 경로를 한 경로로 만드는 과정 -------- 16

[그림 3.6] Type1의 질의 처리 과정 ---------- 17

[그림 3.7] Type2의 질의 처리 과정 ---------- 18

[그림 3.8] Type3의 질의 처리 과정 ---------- 19

[그림 3.9] Type4의 질의 처리 과정 ---------- 20

[그림 3.10] “CXquery 처리기” 모듈에서 처리과정 ---------- 21

[그림 4.1] Document View ---------- 26

[그림 4.2] Path View ---------- 27

[그림 4.3] Tree View ---------- 28

[그림 4.4] Run ---------- 30

[그림 4.5] 필터링을 수행한 후 나오는 결과파일 ---------- 31

[그림 4.6] 예제 PPTL 파일 ---------- 32

[그림 4.7] TMT를 넘어선 데이터를 질의에 사용한 경우 ---------- 34

[그림 5.1] 성능평가 ---------- 40

표 목 차

<표 3.1> CXquery 질의 분류 ---------- 15

<표 4.1> 시스템 구현 환경 ---------- 23

<표 4.2> CYFilter 시스템 메뉴 설명 표 ---------- 24

<표 4.3> 문서 편집 기능 함수 설명 ---------- 28

<표 4.4> 필터링 기능 함수 설명 ---------- 28

<표 5.1> 실험 데이터 명세 ---------- 37

<표 5.2 > C1 (YFilter) 사용 질의문 ---------- 38

<표 5.3> C2 (CYFilter) 사용 질의문 ---------- 38

<표 5.4> 실험 데이터 ---------- 40

<표 5.5> 실험결과 ---------- 40

論 文 槪 要

XML(Extensible Markup Language)은 쉽고 일관된 방법으로 데이터를 포맷하고 전송하는데 사용한다. XML은 그 유연성 때문에 웹 상에서 데이터와 각종 문서 교환을 위한 표준으로 간주되고 있다. 분산 환경에서 스트림 데이터들도 XML을 이용하여 표현되고 있어 이러한 XML 스트리밍 데이터 필터링에 대한 연구가 활발히 이루어지고 있다.

종래에 연구되고 있는 스트리밍 데이터 필터링 시스템들은 사용자가 문서의 구조를 알아야 하는 XPath, XQuery 등의 질의 언어를 사용한다. 위에 제시된 질의 언어를 사용하여 XML 문서의 데이터를 질의하기 위해서는 사용자들은 XML문서의 구조를 알아야 한다. 즉, 사용자가 문서의 루트에서부터 데이터가 있는 곳까지의 경로를 명시하여 질의를 하여야 한다. 그리고, 같은 질의의 내용일지라도 XML 문서의 DTD가 다르게 되면 사용자는 상이한 DTD에 대해서 각각 다른 질의문을 작성하여야 한다.

현재 행해지고 있는 스트리밍 데이터 필터링 시스템의 경로 기반 형식의 질의의 불편함을 없애기 위하여 CXquery의 개념을 도입하여 사용자가 XML 문서의 구조를 고려하지 않아도 스트리밍 XML 데이터를 효과적으로 필터링할 수 있는 스트리밍 XML 필터링 시스템을 구현하였다. 사용자가 CXquery와 스트리밍 XML 데이터를 입력값으로 주면, 시스템은 입력된 XML 문서들의 구조를 파악하여 CXquery를 XPath 기반의 질의로 바꾸어 준 후, XPath 필터기에 넘겨 주어 필터링을 수행하도록 한다.

본 연구에서 제안하고 있는 스트리밍 XML 데이터 필터링은 사용자가 문서의 구조를 모르고도 검색하고자 하는 태그의 이름과 조건식의 비교값만을 주어 결과를 얻을 수 있으므로 문서의 구조를 모르는 사용자 및 XML 문서를 처음 다루어 경로 기반의 질의를 모르는 사용자에게도 도움이 될 것으로 기대된다.

Ⅰ. 서 론

1.1 연구 배경 및 목적

XML(Extensible Markup Language)[1]은 쉽고 일관된 방법으로 데이터를 포맷하고 전송하는 데 사용한다. XML은 그 유연성 때문에 웹 상에서 데이터와 각종 문서 교환을 위한 표준으로 간주되고 있다. 웹 서비스와 같은 분산 컴퓨팅 환경에서 위와 같은 이유로 데이터들은 활발히 XML로 인코딩되어 교환되고 있다. 스트리밍 데이터는 저장되어 있는 데이터가 아닌 계속적으로 빠르게 일시적으로 지나가는 데이터이고 스트리밍 데이터의 형식으로 들어오는 XML 문서를 스트리밍 XML 데이터라고 한다. 네트워크와 같은 환경에서 센서로부터 실시간으로 전송되는 데이터를 처리하는 문제나 네트워크에서 데이터를 모니터링 하는 문제 등 많은 분야에서 이러한 스트리밍 데이터를 사용하는 애플리케이션을 필요로 하고 있다. 스트리밍 데이터는 빠르고, 언제 들어올지 예측할 수 없으며, 계속적으로 한계없이 들어오는 특징을 가진다[2,3].

이러한 특징을 지니는 스트리밍 XML 데이터의 스트리밍 XML 데이터 필터링에 대한 연구가 활발히 이루어지고 있다. 스트리밍 XML 데이터 필터링 시스템에서는 원하는 데이터를 얻기 위해서 사용자가 질의를 등록하면 이 질의들은 서버에 저장되고 서버에 스트리밍 데이터들이 연속적으로 들어오게 된다. 스트리밍 데이터에 대해서 매치되는 질의가 있다면 해당 질의를 등록한 사용자에게 매치된 데이터를 보내준다[4].

원하는 XML 데이터를 추출하기 위해서 기존의 질의 언어[5,6]를 사용할 때, 사용자는 그 문서의 구조를 알아야 한다. 이러한 사용자의 불편함을 없애기 위해 CXquery라는 새로운 질의 언어가 개발되었다[7,8]. 그러나 이 질의 언어에서

제공하고 있는 질의 처리 기법은 XML 문서를 파싱하여 문서에 있는 데이터(엘리먼트, 애트리뷰트)들에 관한 위치 정보를 기존의 데이터베이스에 저장한 후 질의 처리를 한다. 이러한 방식으로 스트리밍 데이터를 저장하고 처리하기에는 데이터베이스에 저장해야 할 내용이 방대할 뿐 아니라 연속적으로 변화하는 특징을 가지는 스트리밍 데이터에 대해서 적용하기가 어렵다. 본 논문에서는 XML 스트리밍 데이터에 적용할 수 있도록 기존의 CXquery 질의를 처리하는 방법인 데이터베이스에 데이터의 정보를 저장하는 방법 대신 새로운 방법을 적용할 것이다.

1.2 연구 내용

본 연구에서는 질의 표현식에서 문서 구조에 독립적인 CXquery의 개념을 도입하고 이를 위한 질의 처리기는 스트림 데이터에 효과적인 기법을 개발한다.

기존의 질의 언어가 가지고 있는 단점인 사용자가 문서 구조를 알아야 하는 불편함을 없애기 위해 CXquery의 개념을 도입하여, 엘리먼트/애트리뷰트의 이름(이하 태그 이름이라 사용한다.)과 질의문의 조건식에 쓰이는 비교값(이하 조건식의 비교값이라 사용한다.)만으로 표현하도록 하였다. 그리고, XPath 기반의 필터링인 YFilter[9,10,11]에 적용할 수 있도록 CXquery 질의를 XPath 경로 기반의 질의문으로 바꾸어, 필터기에 입력값으로 넣어서 질의 결과를 얻어낼 수 있도록 한다.

본 논문에서 제안하고 있는 시스템은 스트리밍 XML 데이터 필터링 기능과 XML, DTD(Document Type Definition) 문서에 관한 작업을 할 수 있는 문서 작성기의 기능도 추가하였다. 이 시스템을 이용함으로써 사용자는 문서의 구조를 모르고서도 질의할 수 있고, 문서의 조작도 할 수 있도록 한다.

본 논문의 전체적인 구성은 다음과 같다. Ⅱ장에서는 관련연구로 필터링을 위한 질의에 사용되는 XML 질의 표현들과 현재의 XML 필터링 시스템 들에 관한 관련 연구를 소개할 것이고, Ⅲ장에서는 이 논문에서 제시하고 있는 시스템에 대한 소개를

할 것이다. Ⅳ장에서는 Ⅲ장에서의 이론을 바탕으로 실제 구현한 CYFilter 시스템에 대하여 설명할 것이고, Ⅴ장에서는 이 시스템을 바탕으로 한 실험평가에 대해서 서술한 다음 마지막 장에서는 결론을 내도록 한다.

Ⅱ. 관련 기술 및 연구동향

본 장에서는, XML 질의 표현 기법들에 대해서 살펴보고, XML문서를 필터링하는 기법들에 대해 그 특징을 간단히 소개한다.

2.1 XML 질의 표현

2.1.1 XPath

XML 문서의 구조는 계층적인 구조를 가지고 있으며, 트리의 형태로 표현될 수 있다. 이러한 XML의 구조에서 문서에 필요한 정보를 얻기 위해서 XPath[6]라는 질의를 사용한다. XPath 는 XML의 엘리먼트를 트리 노드로 보며, 이 질의는 위치 경로와 문맥 노드로 이루어져 있다. 위치 경로는 XML 문서의 루트에서부터 최종 원하는 엘리먼트까지의 경로를 명시하는 위치 스텝들의 순서열이고, 문맥 노드는 지금 현재 XPath에서 위치하고 있는 노드의 위치를 나타낸다. XPath에서는 XML 문서의 계층적인 구조를 항해하는 방식으로 원하는 데이터가 있는 곳까지의 경로를 모두 써 준다.

2.1.2 CXquery(Chamois XML query Language)

CXquery는 XML 문서를 대상으로 문서 구조에 독립적으로 질의를 할 수 있는 질의 표현식이다. 즉, 문서의 구조를 모르고서도 사용자가 질의를 할 수 있다. 기존의 XML 질의 언어는 문서 구조를 따라 항해하는 방식으로 표현되는 경로 기반으로 질의하는 것에 반해 CXquery는 이러한 경로를 명시하지 않고 검색을 원하는 태그 이름과 조건식의 비교값만을 사용하는 표현방식을 따른다.

CXQuery 기반 질의 처리기는 크게 두 부분으로 나뉜다. 첫 번째 부분은 XML문서를 파싱하여 데이터베이스에 저장하는 부분이고, 두 번째 부분은 CXquery 기반 질의문을 입력으로 받아 이것을 질의 처리하여 XML 문서 형태로 결과를 반환하는 부분이다. 첫 번째 부분에서 XML문서를 파싱할 때, 파서는 파싱하는 동안 각 요소를 엘리먼트/ 애트리뷰트 이름인 노드와 이들의 값으로 나눈다. 각 노드들의 계층 또는 순서 정보와 같은 위치를 알 수 있는 노드 식별자를 부여하여 인덱스 테이블에 나누어 그 정보들을 저장한다. 그렇게 함으로써 CXquery에서 데이터 이름만 주어져도 인덱스 테이블에서 노드의 위치 정보를 찾아 빠르게 질의를 할 수 있도록 한다.

2.2 스트리밍 XML 데이터 필터링

XML이 웹 상에서 문서 교환의 표준이 되면서 방대한 양의 스트리밍 XML 문서를 효율적으로 필터링하도록 하는 시스템에 대한 연구가 있어 왔다. 그 중에서 대표적인 것이 XFilter, YFilter이다.

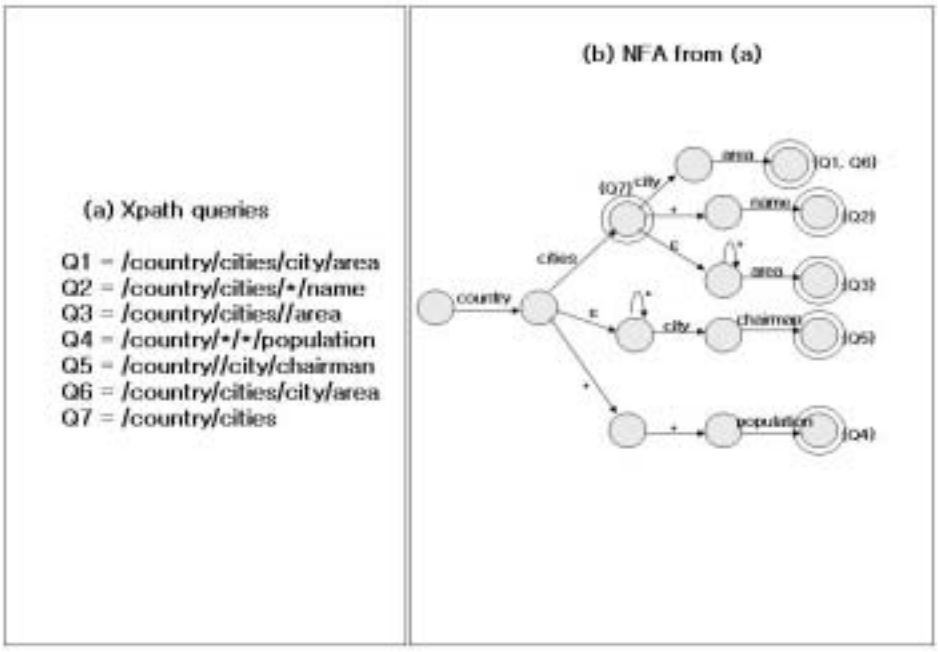
2.2.1 XFilter

XFilter[4,12]는 스트림 XML 데이터를 필터링하는데 있어 XPath 형식으로 표현된 사용자의 프로파일에 기반하여 XML문서를 필터링하는 시스템이다. XFilter 은 스트리밍 XML 데이터 필터링 기법 중에서 초기에 나온 제안방법으로 사용자에 의해서 보내진 XPath 질의를 FSM(Finite State Machines)로 생각하여 처리하는 방식을 사용한다. XFilter는 각각의 모든 질의에 대해 새로운 FSM을 만들어내기 때문에 경로 표현에 있어 공통되는 부분에 대해서 중복적인 작업을 하게 된다.

2.2.2 YFilter

XYFilter에서 질의의 수가 많아질 때 처리량이 감소하는 단점을 보완하기 위해 행해진 연구가 YFilter[9,10,11,13]이다. YFilter는 XPath 경로 형식으로 표현된 질의를 NFA(Nondeterministic Finite Automata)로 표현을 하여 XFilter에서의 중복적인 작업을 없애고, 비결정형 오토마타를 사용함으로써 경로의 공통 부분을 공유하도록 하여 한 번만 처리되도록 한다. 이것은 XFilter에 비해서 표현되는 상태의 수를 줄임으로써 효율적으로 필터링을 수행하도록 하고, 많은 수의 질의 표현에 대해서도 효율적인 필터링을 할 수 있도록 한다.

YFilter의 경로 표현을 NFA로 하여 사용자가 원하는 다수의 질의를 한 개의 머신으로 결합을 한다. 질의의 앞부분(prefix)을 공유하고 있기 때문에 공통되는 엘리먼트에 대해서 질의에 따라 각각 계산을 해 주는 것이 아니라 한 번만 계산해 주면 되기 때문에 XFilter보다 질의의 수가 많을 때 처리량이 좋아지는 장점이 있으며 메모리의 사용량도 적다. [그림 2.1]은 XPath 질의가 비결정형 오토마타의 형식으로 바꾸어진 모습을 나타낸 것이다. 모든 질의에서 공통으로 쓰이는 앞부분(prefix)인 “country”를 공통의 오토마타로 표현하여 각각의 유한 오토마타로 표현할 때보다 나타나는 상태의 수를 줄이도록 하였다.



[그림 2.1] YFilter에서 질의의 NFA표현

현재 스트리밍 XML 데이터 필터링에 대한 연구는 활발하나 이는 모두 사용자가 문서에 대한 구조를 알고 있어야만 질의를 할 수 있는 형태이고, 사용자가 문서의 구조를 모르고도 스트리밍 XML 데이터를 필터링을 하는 방법에 관한 연구는 아직까지 행하여 지지 않고 있다.

Ⅲ. 필터링 시스템 설계

본 장에서는 본 논문이 제안하고 있는 CYFilter 시스템 모델과 그 동작과정을 설명한다. 그리고, 이 시스템에서의 중요 개념인 CXquery를 적용하기 위해 만든 모듈들에 쓰이는 기법에 대해 설명한다.

CYFilter 시스템은 문서 구조에 독립적으로 질의가 가능한 CXquery의 개념을 도입하고, XPath 기반의 필터기인 YFilter를 결합한 시스템이다. 이 시스템에서는 사용자가 XML로 표현된 스트림 데이터를 처리하는데 있어서 XML 문서의 구조를 알지 못하더라도 편리하게 필터링 할 수 있다. XML문서는 반구조화 문서로 데이터들 간에 계층 구조를 형성한다. 그렇기 때문에, XML문서를 질의하기 위해서 사용자는 문서의 구조를 알아야만 한다. 사용자는 문서의 루트로부터 원하는 데이터가 있는 경로까지 명시해 주어야 한다. 이는 사용자가 문서의 구조를 알아야만 하고 비슷한 구조의 상이한 DTD를 가진 문서들에 대해서는 각기 다른 질의문을 던져야 하는 불편함을 주게 된다. 그러나, CXquery를 사용하면 사용자는 문서의 구조를 알지 못하더라도 태그 이름과 조건식의 비교값만을 질의로 던져 원하는 결과를 얻어낼 수 있다.

사용자는 CXquery를 사용함으로써 문서 구조를 알지 못하더라도 질의를 할 수 있고, 그 결과를 얻을 수 있다. XML문서는 사용자의 정의에 의해서 태그가 정해진다. 그래서 비슷한 내용의 문서를 XML로 표현할 때, 작성자에 따라 다양한 XML DTD가 나타나게 된다. 기존의 질의 방법은 사용자가 모든 XML DTD를 알아서 그 구조에 맞추어 질의를 사용하게 된다. 그러나 CXquery를 사용하면 사용자는 단 하나의 질의문으로도 질의가 가능하게 된다. 이는 사용자에게 편리함을 제공할 수 있을 것이다.

이러한 CXquery를 질의어로 사용하기 위해서 본 시스템은 기존의 질의 처리 시스템들과 다른 특징을 가진다. 사용자는 질의하려는 문서의 구조를 고려하지

않아도 되는 대신, 시스템이 문서의 구조를 파악하고 있어야 한다. CYFilter는 필터링 하고자 하는 모든 XML 문서의 구조를 파악하여 사용자로부터 찾으려는 태그 이름과 조건식의 비교값만을 입력 받더라도 그 태그 이름이 들어간 모든 가능한 경로를 찾아내 원하는 결과를 얻어낼 수 있도록 구성하였다.

3.1 질의 시스템 구조

[그림 3.1]은 스트리밍 데이터에 대해 필터링 서비스를 제공하는 CYFilter의 시스템 구조를 나타낸다. CYFilter 시스템은 크게 입력부, 처리부, 출력부로 나누어지고, 처리부는 3개의 모듈로 구성된다. 이는 “경로 생성기”(Path Generator), “CXquery 처리기”(CXquery Processor), “YFilter”로 이루어진다. 처음 두 개의 모듈은 CXquery 질의가 가능하도록 하기 위해 만든 모듈들이고, 세 번째 모듈은 버클리 대학교에서 개발한 YFilter이다. 사용자가 입력부에서 필터링하고자 하는 XML문서(혹은 그 문서가 있는 폴더)와 CXquery 파일을 입력한다. 처리부에서 이 두 정보를 가지고 처리를 하고, 출력부에서 필터링한 결과를 파일로 저장한다.

CYFilter 시스템 동작 과정에 대해서 살펴보면 Step1에서 Step4로 나타낼 수 있다.

Step1. 시스템의 입력값으로 XML문서들과 CXquery 파일이 들어간다.

Step2. CXquery 질의 처리기로 가서 XPath 필터기의 입력 질의 파일 형태인 XPath 질의 파일로 바꾼다.

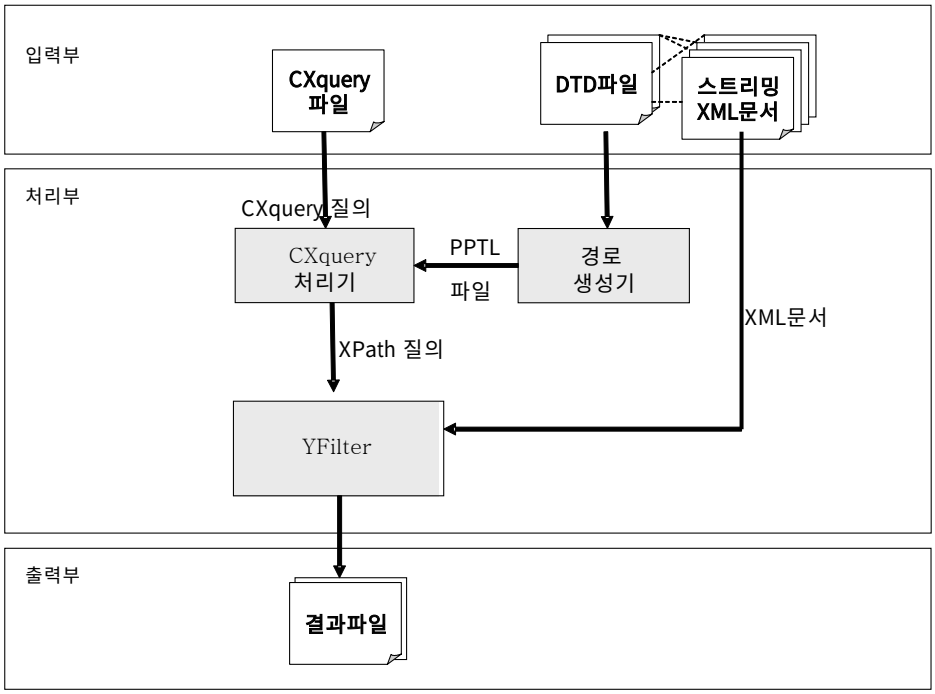
ⅰ) XML문서의 DTD를 경로 생성기를 거쳐 PPTL 파일을 생성한다.

ii) PPTL 파일들과 CXquery 파일에 쓰이 태그 이름을 이용하여 XPath 질의로 변환한다.

Step3. CXquery 질의 처리기에서 변환된 XPath 질의 파일과 스트리밍 XML

데이터를 YFilter에 입력값으로 넣는다.

Step4. YFilter에서 필터링을 실행하여 결과 내용을 결과 파일에 저장하여 사용자에게 알린다.



[그림 3.1] CYFilter 구조도

[용어 3.1] PPTL (Possible Path To Leaf)

DTD 문서가 주어졌을 때, 그 문서에서 나올 수 있는 모든 가능한 경로들을 PPTL이라 정의한다. 이는 루트로부터 리프노드까지 이어지는 전체경로이다.

[용어 3.2] PPTL 파일 (Possible Path To Leaf 파일)

한 DTD 문서에 대해서 모든 PPTL의 집합을 모아놓은 파일을 PPTL 파일이라 정의한다. DTD 한 개당 하나의 PPTL 파일이 생성되며, path_*DTD명*.txt라고 저장된다.

사용자가 사용하는 CXquery를 XPath 질의로 바꾸기 위해서 DTD 문서에 대해서 리

프노드까지의 모든 가능한 경로가 필요하다. 이러한 경로들을 만들어내기 위해서 PPTL과 PPTL 파일을 정의한다. PPTL 파일의 예제는 3.2.1의 [그림 3.4]에 보여진다.

3. 2 시스템 처리부

본 논문에서 제시하고 있는 필터링 시스템에서 가장 핵심 부분인 처리부는 “경로 생성기”, “CXquery 처리기”, XPath 필터기인 “YFilter”로 나누어진다. YFilter가 XPath 기반의 필터기이기 때문에 CXquery를 XPath로 바꾸기 위한 처리 과정이 필요하다. 이를 위해 경로 생성기 모듈과 CXquery 처리기 모듈을 두어 CXquery를 XPath로 변환하도록 하였다. 시스템의 처리부에서 쓰이고 있는 3가지 모듈들에 대해서, 각각의 모듈이 왜 필요한지, 그리고 그것들의 동작 기법에 관해서 서술할 것이다.

3.2.1 경로 생성기(Path Generator)

경로 생성기는 입력 XML 문서의 DTD를 파악하여, 각 DTD에 대해 PPTL(Possible Path To Leaf)들을 생성해 내는 부분이다. 경로 생성기를 거치면 한 DTD 문서의 PPTL들이 저장되어 있는 PPTL 파일(Possible Path To Leaf 파일)이 생성된다. CXquery 처리기에서 입력받은 CXquery 파일을 XPath로 된 질의 파일로 생성할 때, XML 문서의 PPTL 파일이 필요하게 된다. 이 때 필요한 경로를 생성하기 위해 만든 모듈이 경로 생성기이다.

CYFilter 시스템 입력 값의 하나로 필터링할 XML 문서 혹은 문서가 있는 폴더 이름(이 경우 다수의 XML문서가 입력값이 된다.)이 쓰인다. 그러면 경로 생성기는 XML 문서의 머리 부분에서 DTD 파일 이름을 알아내고, 그 DTD 문서들을 스캔한다. 한 DTD 파일의 PPTL들을 생성하고, 그것들은 PPTL 파일에 저장된다. 경로 생성기를 거치게 되면 한 DTD 문서 당 한 개의 PPTL 파일이 생성되고, 후에 DTD 문서의 PPTL

파일이 필요할 때, 기존의 PPTL 파일들을 재사용한다.

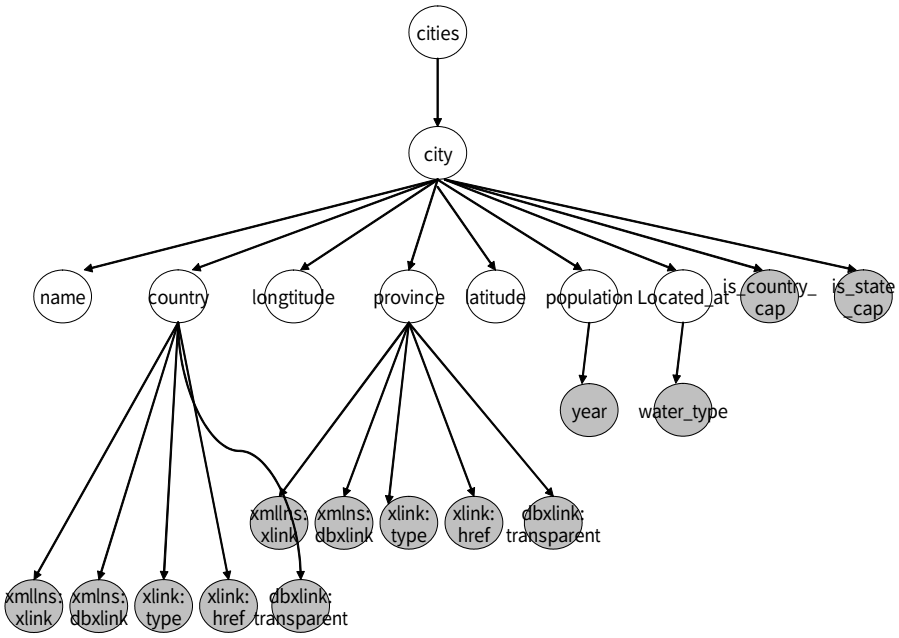
[그림 3.2]는 경로 생성기에서 쓰이는 경로 생성 알고리즘을 나타낸 것이다.

```
while(필터링할 문서 존재) {
  checkDTD() {
    line by line XML 문서 읽음
    If( "<!DOCTYPE" 로 된 라인) {
      라인의 SYSTEM 다음에 나오는 DTD 파일명 알아냄
      ReadXML(DTD파일명) 호출
    }
  }
}
ReadXML(DTD파일명) {
  Do {
    IF(태그 이름 타입 == 엘리먼트) {
       부모 엘리먼트에 자식 엘리먼트를 추가한다.
       ex) //parent/child
    }
    ELSE IF(태그 이름 타입 == 애트리뷰트) {
        "<!ATTLIST"에 나온 엘리먼트에 애트리뷰트의 내용을 추가한다.
       한 엘리먼트에 여러 애트리뷰트가 선언될 수 있는데, 이 때에는
       그 엘리먼트 뒤에 이어 적는다.
       ex) //elem[@attr1][@attr2]
   }
  } While(!EOF)
}
```

[그림 3.2] 경로 생성 알고리즘

XML문서를 읽으면 문서의 머리에 "<!DOCTYPE *root_element* SYSTEM "*dtd 파일명.dtd*">" DTD 선언부가 나온다. 여기에서 DTD 파일명을 찾아내고 그 DTD 문서를 스캔해 나가면서 경로를 생성한다.

[그림 3.3]은 예제 XML문서를 DOM 트리로 표현한 그림이다. 색칠하지 않은 타원형은 엘리먼트를 표현한 것이고, 색이 칠해져 있는 타원형은 애트리뷰트이다.



[그림 3. 3] 예제 XML문서의 DOM 트리

[그림 3.3] XML 문서의 DTD 파일이 [그림3.2]에 쓰인 경로 생성 알고리즘을 거치면 [그림 3.4]와 같은 7개의 경로를 갖는 DTD 문서의 PPTL 파일이 생성된다.

```
/cities/city[@is_country_cap][@is_state_cap]/name
/cities/city[@is_country_cap][@is_state_cap]/country[@xmlns:xlink][@xmlns:dbx
                    link][@xlink:type][@xlink:href][@dbxlink:transparent]
/cities/city[@is_country_cap][@is_state_cap]/longtitude
/cities/city[@is_country_cap][@is_state_cap]/province[@xmlns:xlink][@xmlns:db
xlink][@xlink:type][@xlink:href][@dbxlink:transparent]
/cities/city[@is_country_cap][@is_state_cap]/latitude
```

```
/cities/city[@is_country_cap][@is_state_cap]/population[@year]
/cities/city[@is_country_cap][@is_state_cap]/located_at[@water_type]
```

[그림3.4] [그림 3.3]의 PPTL 파일

3.2.2 CXquery 처리기

CYFilter 시스템은 XPath 기반의 필터기인 YFilter를 사용한다. 그러므로, CYFilter 시스템의 입력값으로 입력 받은 CXquery 파일을 XPath 질의 파일로 바꾸는 과정이 필요하다. CXquery 처리기는 입력으로 들어온 XML문서의 DTD 문서의 PPTL 파일들과 CXquery 파일을 가지고 XPath 질의 파일로 변환해 주는 모듈이다.

CXquery 처리기에서는 DTD파일의 PPTL 파일들과 CXquery 파일이 필요하다. DTD 문서의 PPTL 파일이 있으면 그것들을 사용하고, 없으면 경로 생성기를 거쳐서 새 PPTL 파일을 생성한다. CXquery 처리기는 입력 DTD의 PPTL 파일들을 차례대로 읽어 나가면서, CXquery 파일에 명시된 태그 이름이 존재하는 경로들을 뽑아 낸다. 만약, 질의문에 조건식의 비교값이 있는 경우 각 경로에 값을 넣도록 하여 그 조건에 맞는 결과를 찾아낼 수 있도록 한다. CXquery 파일이 XPath 질의 파일로 변환된 내용은 “xpath_*CXquery파일명*.txt”에 적는다. CXquery 파일 한 개당 한 개의 “xpath_*CXquery파일명*.txt”을 생성하여 후에 같은 질의를 사용하고자 할 때, 재사용될 수 있도록 한다.

CXquery의 질의 표현유형에는 조건식의 개수에 따라, 조건식의 비교값의 존재 여부에 따라 분류할 수 있다. 조건식의 개수에 따른 분류는 단일 조건식을 사용하는 경우, 복수 조건식을 사용하는 경우로 나뉜다. 복수 조건식을 사용하는 경우에는 또 다시 and 논리 연산자를 사용하는 경우와 or 논리 연산자를 사용하는 경우로 나뉘어 조건식의 개수에 따른 분류는 총 3개로 분류된다. 조건식 비교 값의 존재 여부에 따라서 2가지로 분류된다.

<표 3.1> CXquery 질의 분류

분류 기준			CXquery 예
조건식의 개수에 따른 분류	단일 조건식	Type1: and, or 논리 연산자가 없는 질의문	name
	복수 조건식	Type2: and 논리 연산자 질의문	latitude **and** name
		Type3: or 논리 연산자 질의문	latitude **or** name
조건식 비교값의 유무	Type4: 조건식에 비교 값이 있는 경우		name = “SEOUL”
	Type5: 조건식에 비교 값이 없는 경우		name

<표3.1>에서 Type1의 경우, PPTL 파일을 처음부터 읽어가면서, CXquery에 쓰인 “name”이 있는 경로들을 뽑아낸다.

CXquery에 Type2는 두 가지 경우를 고려하여 구현한다. 첫 번째는 한 경로에 태그 이름 “latitude”와 “name”이 모두 있을 때, 그 경로를 뽑아 내는 경우이고, 두 번째는 한 경로에 두 태그를 모두 가지고 있지는 않지만, 한 PPTL 파일 안에 “latitude”를 갖는 경로와 “name”을 갖는 경로가 모두 포함할 때, 그 두 경로를 뽑아내는 경우이다. 첫 번째 경우는 한 경로를 써주면 되나, 두 번째 경우는 뽑아낸 두 개의 경로를 가지고 두 태그가 모두 들어있는 한 개의 경로로 만들어 주는 추가적인 작업을 해 주어야 한다. 먼저, 두 개의 경로에서 공통 경로까지 쓴다. CXquery에서 공통 경로를 제외하여 첫 번째 태그가 있는 곳까지의 경로를 곽괄호(“[]”)안에 쓴다. 그리고, 두 번째 태그는 공통 경로를 제외한 나머지의 경로를 위에서 만들어진 경로 뒤에 이어 쓴다. 이의 과정은 아래 [그림 3.5]에 설명한다.

Step1

사용자의 입력: 논리 연산자 **and**가 있는 질의문(Q1), 필터링할 XML 문서

Q1: **name** and **latitude**

Step2

입력으로 들어온 XML의 DTD 문서의 PPTL 파일을 읽으면서 질의문에 만족하는 경로를 찾는다.

예. 위의 [그림 3.4]의 PPTL 파일에서 Q1 에 만족하는 경로는 다음과 같다.

/cities/city[@is_country_cap][@is_state_cap]/name

/cities/city[@is_country_cap][@is_state_cap]/latitude

Step3

두 경로의 공통 경로를 뽑는다.

예. **/cities/city[@is_country_cap][@is_state_cap]**

Step4

위의 두 경로 중 첫 번째 경로에서 공통 경로를 뺀 나머지 경로를 “[ ]” 안에 넣고 Step3의 공통 경로 뒤에 이어 쓴다.

예. /cities/city[@is_country_cap][@is_state_cap]/**name**에서 공통 경로 뺀 나머지 경로 **name**을 “[]”안에 쓴다.

/cities/city[@is_country_cap][@is_state_cap] **[name]**

Step5

위의 두 경로 중 두 번째 경로에서 공통 경로를 뺀 나머지 경로를 Step4 에서 만들어진 경로 뒤에 쓴다.

예. /cities/city[@is_country_cap][@is_state_cap]/**latitude**에서 공통 경로를 제외한 경로 latitude

/cities/city [@is_country_cap][@is_state_cap] [name] **/latitude**

위의 Step1에서 Step5을 거쳐 한 개의 XPath 경로가 생긴다.

/cities/city[@is_country_cap][@is_state_cap][name]/latitude

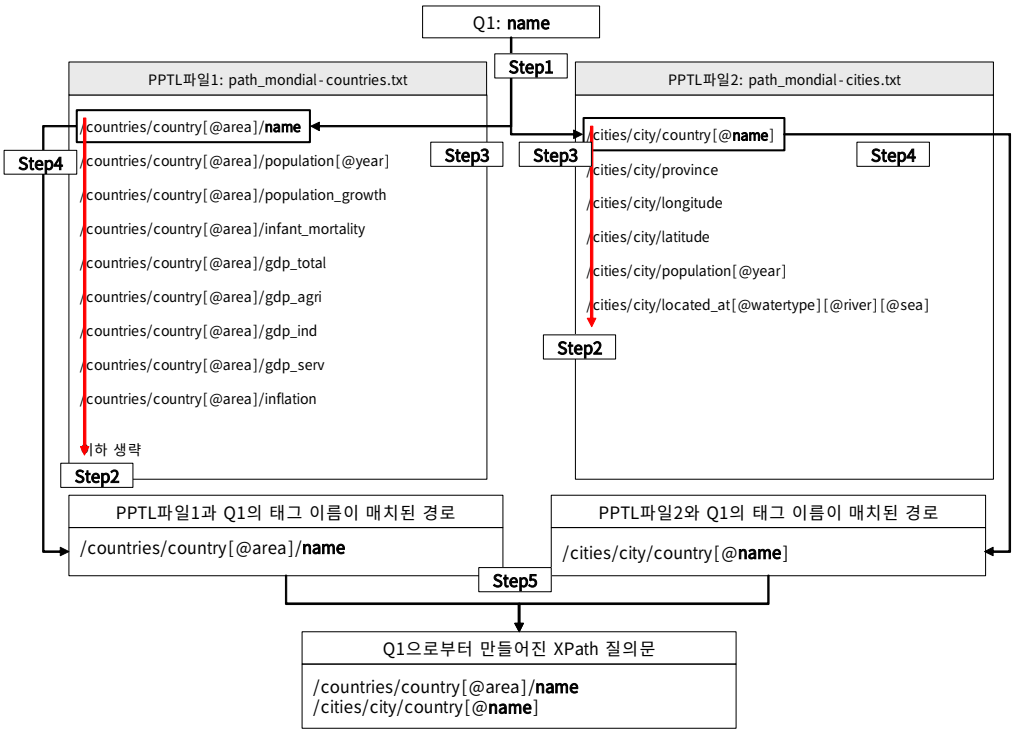
[그림 3.5] 논리 연산자 and 사용시, 두 경로를 한 경로로 만드는 과정

Type3의 경우는 PPTL에 “river”, “sea” 둘 중 하나라도 있는 경우, 그 경로들을 뽑아 낸다.

[그림 3.6], [그림 3.7], [그림 3.8]은 Type1, Type2, Type3에 따른 질의

생성과정을 예제로 나타낸 것이다.

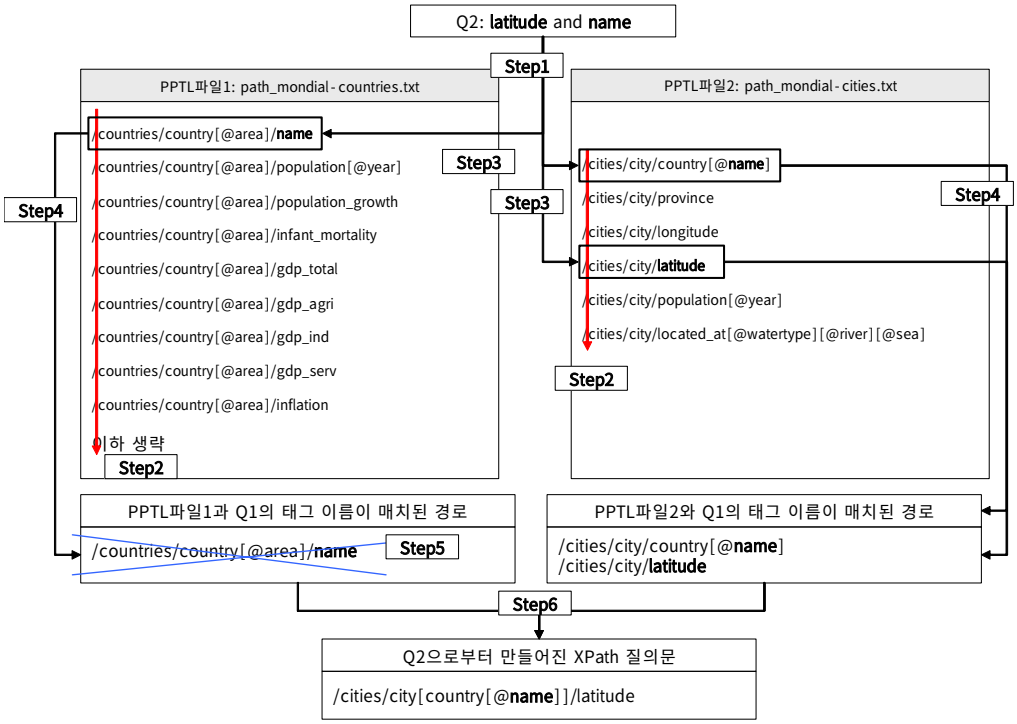
[그림 3.6]은 CXquery 질의문이 태그 이름 “name”만 주어진 경우로, CXquery에 단일 조건식이 사용되는 경우의 처리 과정을 나타낸 것이다. XML 문서들의 PPTL 파일들을 스캔하면서 태그 이름이 일치하는 경로만을 추출한다.



Step1 입력값으로 질의문 Q1이 들어온다.
Step2 PPTL 파일들을 스캔해 나간다.
Step3 PPTL 파일을 스캔하면서, Q1에 쓰인 태그 이름을 포함하는 경로를 찾아낸다.
Step4 위의 경로를 뽑아낸다.
Step5 다수의 PPTL 파일들로부터 뽑아낸 경로들을 모은 것이 변환된 XPath이다.

[그림 3.6] Type1의 질의 처리 과정

[그림 3.7]은 CXquery문에 “and” 논리 연산자가 있을 때, 처리를 나타낸다. CXquery문 Q2가 주어진 경우로, PPTL 파일들을 스캔해 나가면서 한 경로에 두 태그 이름이 있는 경우나 한 파일 내에 두 태그 이름이 존재하는 경우를 검사한다.



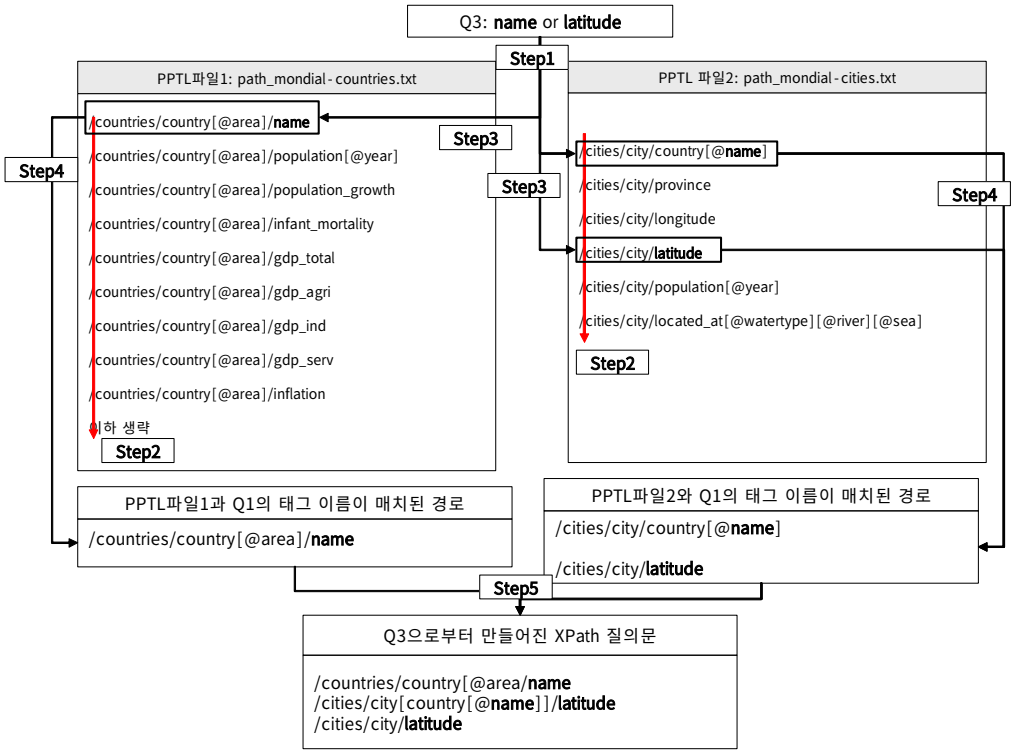
Step1 입력값으로 질의문 Q2가 들어온다.
Step2 PPTL 파일들을 스캔해 나간다.
Step3 PPTL 파일을 스캔하면서, “latitude”나 “name”을 포함하는 경로를 찾아낸다.
Step4 위의 경로를 뽑아낸다.
Step5 PPTL로부터 뽑아낸 경로에서 한 경로에 두 태그 이름이 있거나, 한 파일 내에 두 태그 이름이 들어갔는지를 검사한다. 두 조건이 아닌 경우에는 뽑아낸 경로를 지운다.
Step6 다수의 PPTL 파일들로부터 뽑아낸 경로들을 모은 것이 변환된 XPath이다.

[그림 3.7] Type2의 질의 처리 과정

PPTL파일1로부터 태그 이름이 들어간 경로는 그 경로 파일 내에 다른 태그 이름인

“latitude”가 없으므로 최종 경로에 들어갈 수 없다. 그러나, PPTL파일2로부터 태그 이름이 들어간 경로들은 같은 경로 파일 내에 두 태그 “name” 와 “latitude”가 모두 들어 있으므로 이들을 뽑아내어 한 개의 경로로 만들어 준다.

[그림 3.8]은 CXquery문 Q3가 사용된 경우로 “or” 논리 연산자가 사용된 경우의 질의 처리를 나타낸다. 이 경우 PPTL 파일을 스캔하면서 두 태그 이름 중 하나를 포함하고 있으면 경로로 뽑아낸다.



Step1 입력값으로 질의문 Q3가 들어온다.
Step2 PPTL 파일들을 스캔해 나간다.
Step3 PPTL 파일을 스캔하면서, Q1에 쓰인 “name”나 “latitude”를 포함하고 있는 경로를 찾아낸다.
Step4 위의 경로를 뽑아낸다.
Step5 각 PPTL 파일로부터 얻어진 위의 경로들을 모으면 변환된 질의문이 된다.

[그림 3.8] Type3의 질의 처리 과정

<표 3.1>를 보면 CXquery 질의는 조건식의 비교값의 존재 여부에 따른 2가지로 나뉜다. 질의에 조건식의 비교값이 없는 경우 아무런 처리를 하지 않으나 조건식의 비교값이 있는 경우에는 약간의 처리를 해 주어야 한다. CXquery는 문서 구조에 독립적인 질의를 하기 때문에 조건식에 제시된 태그 이름과 조건식의 비교값 사이에 다른 엘리먼트 혹은 애트리뷰트가 존재할 수 있다는 것을 고려해야 한다.

만약 "/cities/city/area/name"의 경로와 질의문 "area=KOREA"가 있는 경우를 생각해 보자. "KOREA"라는 조건식의 비교값은 "area"라는 태그 이름에 대입될 수 있다. 그리고, "area" 하위의 엘리먼트인 "name"에도 대입될 수 있다. CXquery를 이용한 질의 처리기에서는 조건식에 제시된 태그 이름과 조건식의 비교값 사이에 다른 엘리먼트나 애트리뷰트들이 삽입되는 경우를 처리해야 하기 때문에 후자의 경우를 고려해 주어야 한다. 즉, 주어진 태그 이름을 포함하는 하위의 엘리먼트 또는 애트리뷰트에 주어진 조건식의 비교값을 대입해 주어야 한다.

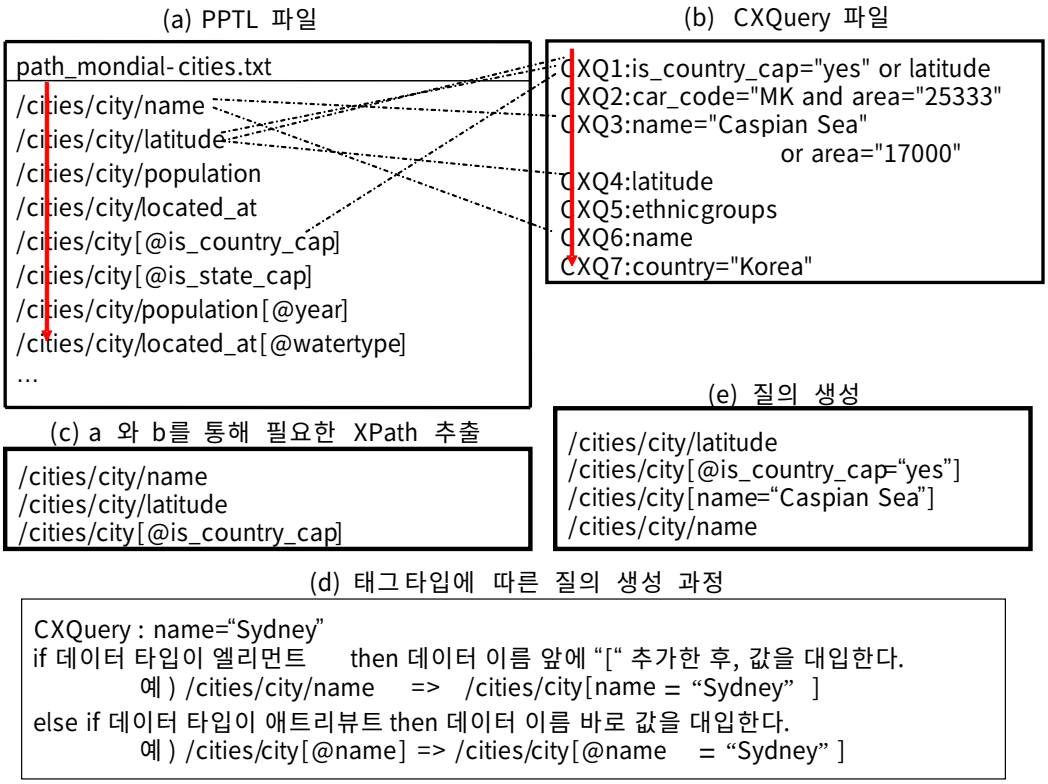
```
Q4: tag_name = value

IF(tag_name 이후에 나타나는 태그의 타입 == ELEMENT)
{
    tag_name 앞 경로를 공통으로 쓰고, a), b)를 추가하여 2개의 경로
    만듦
    /*2개의 XPath 질의가 생긴다.*/
    a) tag_name [text()=value]
    b) tag_name //*[text()=value]
}
ELSE IF(tag_name 이후에 나타나는 태그의 타입 ==ATTRIBUTE)
{
  tag_name 이후에 나타나는 애트리뷰트에 value를 대입하고 value 가 없는
  애트리뷰트는 경로에서 지운다.
```

```
    /*tag_name 이후에 나타나는 애트리뷰트 개수만큼 XPath 질의가 생긴다.*/
     예) /a1/a2/tag_name/a3[@a4][@a5]/a6[@a7]
     - /a1/a2/tag_name/a3[@a4="value"]/a6
     - /a1/a2/tag_name/a3[@a5="value"]/a6
     - /a1/a2/tag_name/a3/a6[@a7="value"]
}
```

[그림 3.9] Type4의 질의 처리 과정

[그림 3.10]은 "CXquery 처리기" 모듈에서의 처리과정을 그림으로 표현한 것이다. CXquery 파일과 XML 문서가 입력으로 들어오면, (a), (b) 단계에서와 같이



[그림 3.10] "CXquery 처리기" 모듈에서 처리과정

시스템은 CXquery와 XML문서의 PPTL 파일들을 스캔한다. (c) 단계에서 CXquery

조건식에 쓰인 태그 이름이 들어간 경로를 PPTL 파일에서 찾고, 그 XPath 경로들을 뽑아낸다. 조건식의 비교값이 주어진 경우 (d) 단계를 거쳐 조건식의 비교값을 (c)에서 뽑아낸 XPath에 대입한다. 이 과정을 모두 거치면 (e)와 같은 질의문이 생성된다.

3.2.3 YFilter

XPath 기반의 필터링 부분은 버클리 대학에서 제안한 YFilter를 바탕으로 하여 구성되었다. CYFilter 시스템의 입력 값은 처리하고자 하는 XML 문서들과 CXquery 질의 파일이다. 그러나 XPath 기반의 필터기인 YFilter는 CXquery질의문을 처리하지 못하기 때문에, CXquery 처리기를 통해 XPath 질의 파일로 변환된 질의 파일을 생성해 주어야 한다. 이렇게 하여 XPath 질의 파일이 생성되면, YFilter는 XPath 질의 파일과 XML 문서를 바탕으로 필터링을 수행한다.

다량의 XML 스트리밍 데이터를 처리하고 그 결과를 보여줘야 하기 때문에 YFilter 와는 달리 필터링 결과를 화면에 보여주지 않고, 결과 파일에 저장하였다. 파일에 저장함으로써 사용자가 결과를 파일에 저장할 수 있도록 하고, 화면에 보여주는 시간보다 필터링된 결과를 더 빨리 얻을 수 있도록 하였다.

Ⅳ. CYFilter 시스템 구현

본 장에서는 사용자가 문서의 구조를 고려하지 않고도 필터링할 수 있는 CYFilter 시스템의 구현환경과 구현 결과에 대해 살펴본다. CXquery 파일과 XML 문서들이 입력으로 들어오게 되면 CYFilter는 먼저 CXquery 파일을 XPath 질의 파일 형태로 변환하고, 이것과 XML 문서를 XPath 질의 기반인 YFilter에 통과시켜 결과를 파일로 얻도록 한다.

본 구현에서는 결과값들을 보여주는 대신 그것들을 파일로 저장하게 함으로써 사용자에게 결과를 리턴해 주는 시간을 줄여 준다. GUI화면으로 필터링을 할 수 있도록 하여 사용자에게 시각화를 제공한다. 그리고, CXquery를 이용함으로써 사용자가 문서의 구조를 알지 못하더라도 필터링을 할 수 있는 편리함을 제공한다.

4.1 시스템 구현 환경

본 논문에서 구현한 CYFilter 시스템의 구현 환경은 [표 4.1]과 같다.

<표 4.1> 시스템 구현 환경

운영 체제	Windows XP Professional
XML 스트리밍 필터기	YFilter 1.0
개발 도구 및 언어	JDK 1.4, Ultra Edit-32, JBuilder 6

본 XML 스트리밍 데이터 필터링은 Microsoft Windows XP Professional 환경을 기반으로 구현하였다. 편집기와 결과 가시화 부분은 JBuilder 6을, CXquery 질의

파일을 처리하기 위한 과정은 JDK1.4와 Ultra Edit-32 Text Editor를 사용하여 Java로 구현하였다.

4.2 시스템 구성

CYFilter 시스템은 XML, DTD 문서들의 편집 기능을 제공하고, CXquery 기반의 질의를 필터링하는 시스템이다.

<표 4.2> CYFilter 시스템 메뉴 설명 표

기능	메뉴	부메뉴	기능
문서 편집 기능	File	XML File (Open, Save)	XML 문서의 내용을 열고, 저장
		DTD File (Open, Save)	DTD 문서의 내용을 열고, 저장
		EXIT	CYFilter 종료
	View	Document View	XML, DTD 문서의 내용을 보여줌
		Tree View	DTD 문서의 트리 구조를 보여줌
		Path View	DTD 문서의 PP 파일을 보여줌
		Clear Display	화면 지움
필터링 기능	Generator	Path Generator	DTD 문서의 PP 파일 생성
	Run	Filter	필터링 수행

4.2.1 문서 편집 기능

CYFilter의 문서 편집은 <표 4.2>에 나온 메뉴 중 File, View와 관련한 기능으로 XML, DTD 문서에 대한 편집 기능과 뷰 기능을 제공한다. 다음의 <표 4.3>은 CYFilter의 문서 편집 기능에 쓰이는 함수를 정리한 표이다.

<표 4.3> 문서 편집 기능 함수 설명

메뉴	함수	설명
File	show_ofile()	XML, DTD 문서 내용을 보여주는 함수
	jMenuFileDTDSave_actionPerformed()	수정된 DTD를 저장하 는 함수
	jMenuFileXMLSave_actionPerformed()	수정된 XML을 저장하는 함수
View	show_tree()	DTD 문서를 트리 구조로 보여주는 함수
	make_tree	루트 정보를 가지고 트리 생성하는 생성자
	show_file()	PP 파일을 보여주는 함수

4.2.1.1 File

File 메뉴는 XML 파일, DTD 파일들을 편집할 수 있도록 하였다.

- XML FILE: 서브 메뉴 FILE Open, FILE Save가 있어 XML 문서에 대해 파일을 열고, 수정한 후에 저장할 수 있도록 하였다.
- DTD FILE: 서브 메뉴 FILE Open, FILE Save가 있어 DTD 문서에 대해 파일을 열고, 수정한 후에 저장할 수 있도록 하였다.

▪ EXIT: CYFilter 종료하고자 할 때 사용한다.

4.2.1.2 View

View 부는 문서의 내용을 보여주는 기능을 한다. 아래는 View 메뉴에 나오는 기능들에 대해 서술한 것이다.

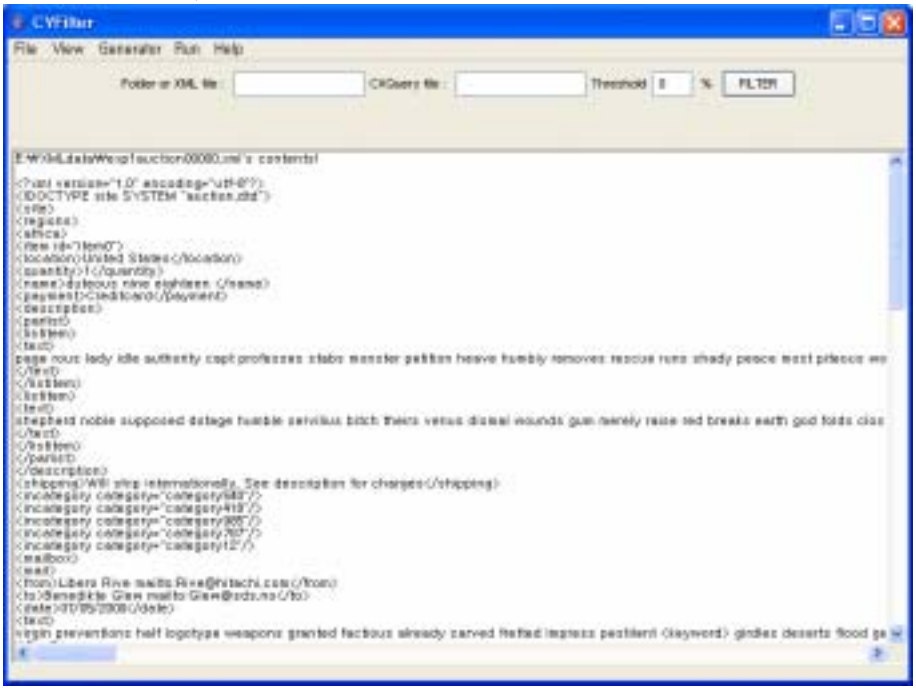
▪ Document View: XML, DTD 문서의 내용을 보여준다.

▪ Path View: DTD 문서를 보고, PP 문서의 내용 즉, DTD 문서의 모든 가능한 경로들을 보여준다.

▪ Tree View: DTD 문서의 내용을 트리 형태로 보여준다.

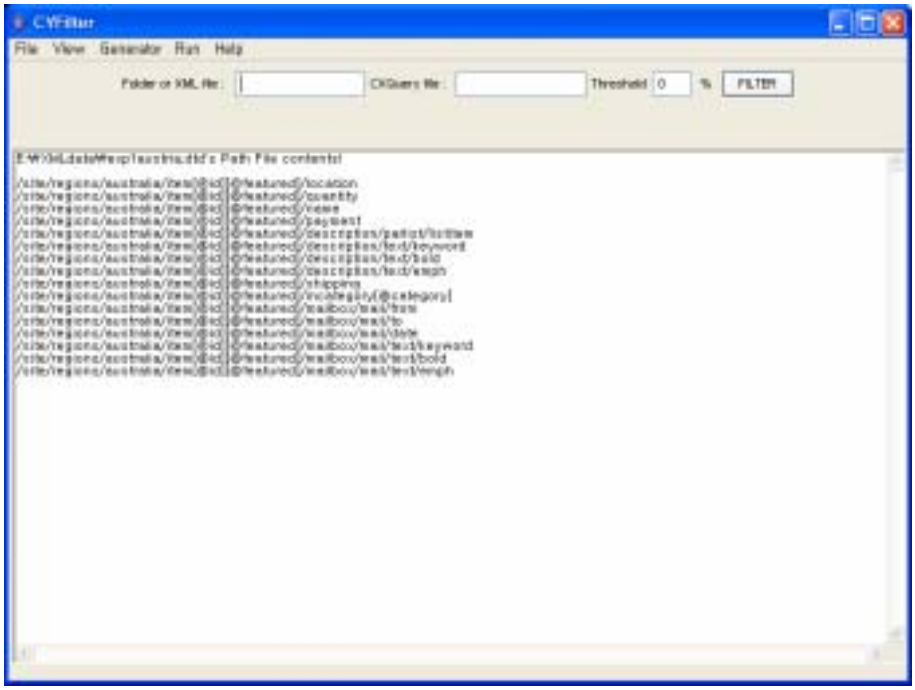
▪ Clear Display: 화면을 지워주는 메뉴이다.

Document View는 원하는 문서를 선택하여 GUI의 결과를 보여주는 창에 문서의 내용을 보여준다. [그림 4.1]은 Document View를 실행한 모습이다.



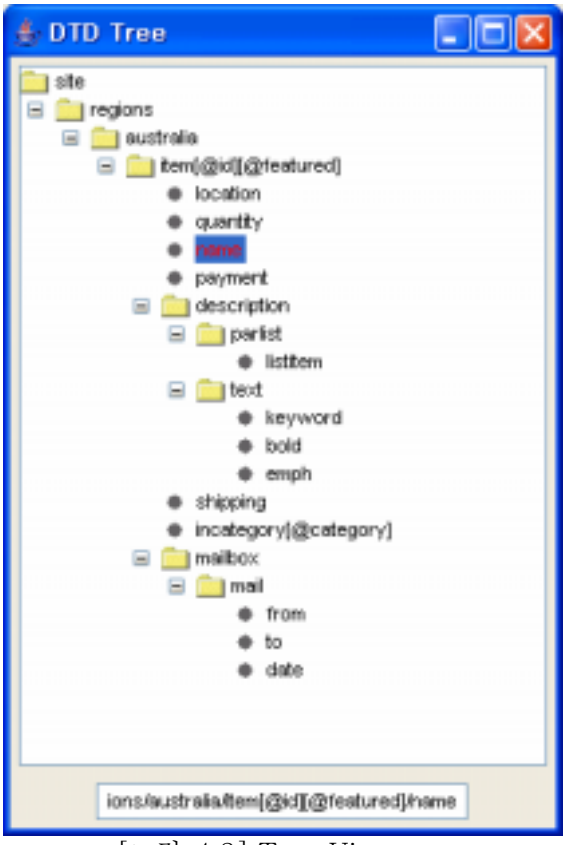
[그림 4.1] Document View

Path View는 DTD 문서의 PPTL 파일의 내용을 보여준다. 이는 [그림 4.2]과 같이 나타난다.



[그림 4.2] Path View

Tree View는 사용자가 원하는 DTD 문서의 트리 형태를 볼 수 있다. DTD 문서에 대한 파일 다이알로그 창이 떠서, DTD 문서를 선택할 수 있도록 한다. [그림 4.3]는 Tree View의 실행 모습이다..



[그림 4.3] Tree View

4.2.2 필터링 기능

CYFilter의 필터링 기능은 <표 4.2> 중 Generator, Run과 관련한 기능이다. 필터링을 위해 PPTL 파일을 만드는 Generator와 실제 필터링을 수행하는 Run과 빠른 실행의 필터링을 하는 부분이 필터링 기능에 속한다. <표 4.4>는 필터링 기능에 쓰이는 함수들을 표로 나타낸 것이다.

<표 4.4> 필터링 기능 함수 설명

메뉴	함수	설명
Generator	CheckDTD()	XML 문서의 DTD를 알아내는 함수

	Read_XML	XML의 DTD 문서를 처음부터 읽으면서 경로를 생성하는 함수
	know_position	애트리뷰트일 때 경로에서 처리하는 함수
Run	rec_name()	CXQuery 질의 파일을 XPath 질의 파일로 만드는 함수
	valid_filter()	경로 생성시 TMT를 넘어서는지 확인하는 함수
	show_word()	TMT를 넘었을 때 다른 대체할 데이터 이름을 보여주는 함수
	addingqry()	CXquery 질의에 연산자가 없을 때 XPath 질의를 함수
	making_qry1()	CXQuery 질의에 and 연산자가 있을 때 XPath 함수
	making_qry2()	CXquery 질의에 or 연산자가 있을 때 XPath 만드는 함수
	ill_attr()	최종 XPath 질의에서 값이 없는 애트리뷰트를 지워주는 함수
	filter_result()	XPath 질의와 XML 문서를 가지고 필터링을 수행하는 함수

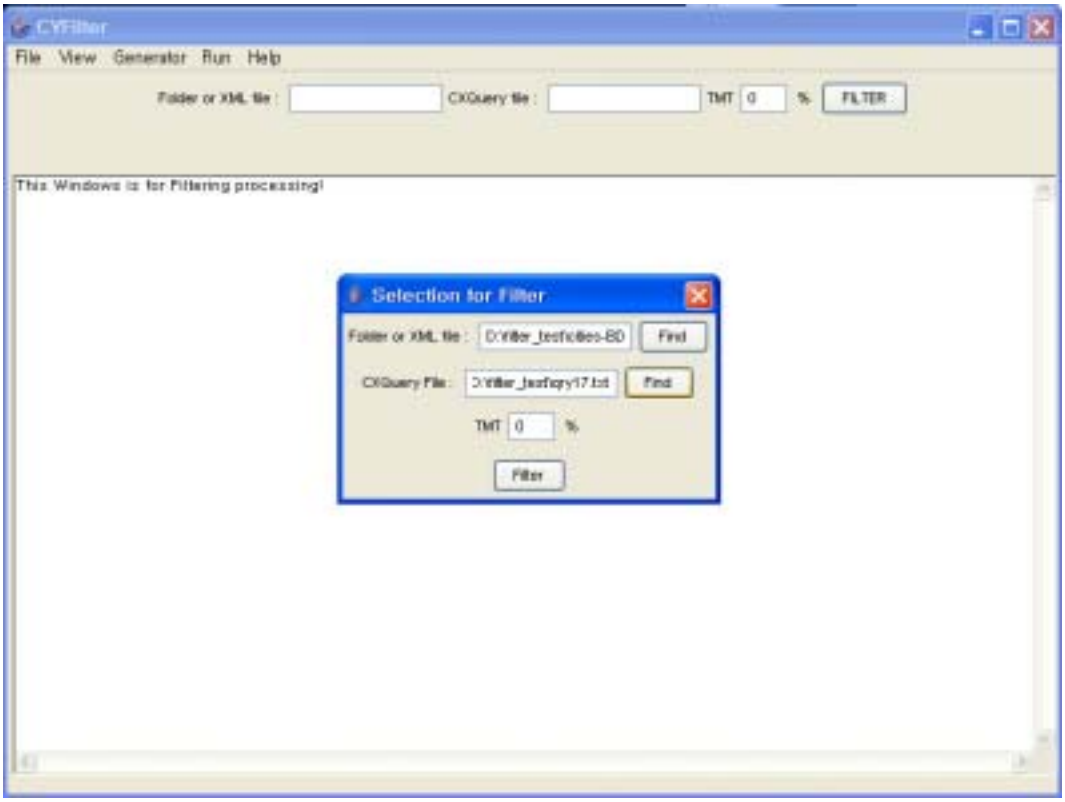
4.2.2.1 Generator

이는 사용자가 DTD 문서를 선택하면, 문서의 가능한 경로를 만들어 PPTL 파일에 저장한다. 이것은 CYFilter에서 경로 생성기 부분에서 DTD 문서의

PPTL파일을 만들어 주는 기능이다.

4.2.2.2 Run

이것은 필터링을 수행하도록 한다. XML 파일 선택과 CXquery 파일 선택을 하도록 하는 파일 선택창에서 사용자가 파일 찾기를 눌러 선택할 수 있도록 한다.



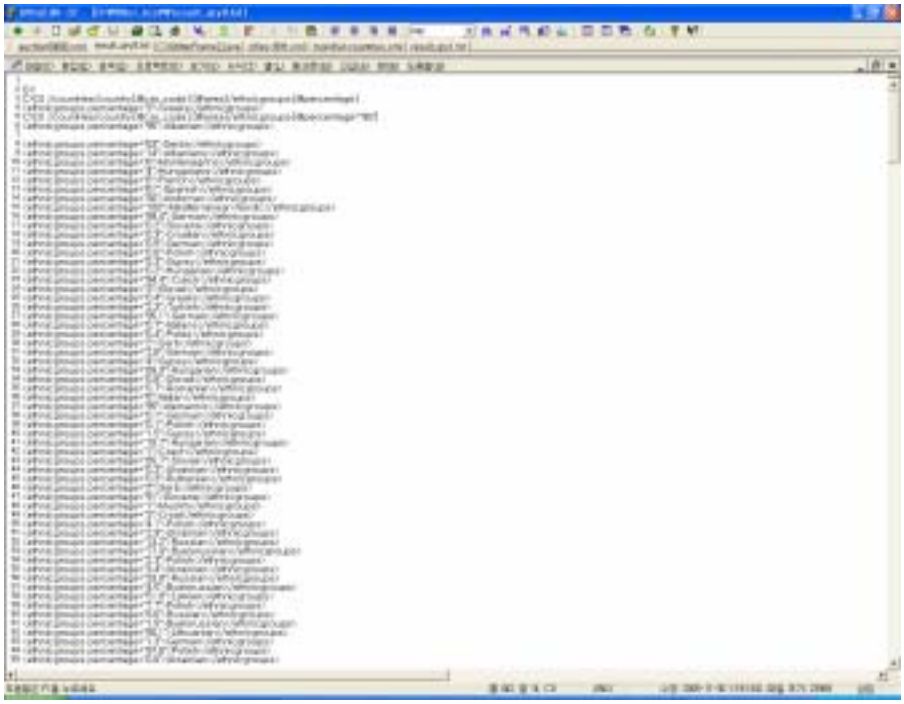
[그림 4.4] Run

4.2.2.3 빠른 실행 FILTER

Run의 FILTER 기능과 같다. 그러나, 사용자가 직접 XML문서와 CXquery 파일을 타이핑 하는 것이 RUN에 쓰이는 필터링과 다른 점이다.

[그림 4.5]은 CYFilter 시스템을 실행하였을 때 질의 조건에 맞는 XML 문서가

필터링되어 결과 파일에 저장된 그림이다.



[그림 4.5] 필터링을 수행한 후 나오는 결과파일

4.2.3 TMP, TMR, TMT (Tag Match Path, Tag Match Ratio, Tag Match Threshold)

CXquery 파일에서 XPath 질의 파일로 변환할 때, 조건절에 쓰인 태그 이름이 들어간 경로들을 뽑아내는 과정에서 태그 이름이 상위에 있게 되면 전체 경로에서 나타나는 비율이 높게 된다. 이러한 경우 사용자의 의도를 넘은 넓은 범위의 결과를 보여주게 된다. 그리고 많은 수의 XPath 질의가 생겨 결과를 얻는 시간도 많이 걸리게 된다. 이에 본 절에서는 사용자의 의도를 넘어서지 않는 결과를 얻을 수 있고 처리 시간도 줄이기 위한 개념인 TMP, TMR, TMT를 정의한다.

[용어 4.1] TMP (Tag Match Path)

PPTL 파일에 있는 경로 중에서 태그 이름(엘리먼트/애트리뷰트)과 매치되는 경로들

의 집합을 말한다. 태그 이름 tag_name라고 하는 경우, TMP(tag_name)이라 표현한다.

[용어 4.2] TMR (Tag Match Ratio)

PPTL 파일의 전체 경로 개수에서 태그 이름의 TMP 개수의 비율로 정의한다.

태그 이름 tag_name이 있으면, 이에 대해 TMR(tag_name)이라 표현하고, 수식은 다음과 같다.

$$TMR(tag\_name) = \frac{\text{TMP(tag\_name)의개수}}{PPTL\text{파일의전체경로수}}$$

[용어 4.3] TMT (Tag Match Threshold)

필터링을 수행할 때, 사용자가 지정하는 값으로, 필터링의 수행 여부를 결정하는 값이다. 다수의 CXquery문이 주어지면 다수의 태그 이름이 존재하게 된다. 이 때, 각각의 태그 이름에 대해서 TMP를 찾은 후, TMR을 구한다.

{tag_name1, tag_name2,..., tag_namei}와 같은 태그 이름 집합이 있다고 하자. 각 태그의 TMR 을 구하여 최대 값이 TMT을 넘는지를 검사하여 필터링 수행 여부를 결정한다.

$TMT >= MAX(TMR(tag\_name1), TMR(tag\_name2), ..., TMR(tag\_namei))$ 인 경우에만 필터링을 수행한다.

[용어 4.3]에서 CXquery의 조건식에 쓰인 태그 이름의 최대 TMR이 TMT를 넘어서게 되면 필터링을 수행하지 않는다. [용어 4.1][용어 4.2][용어 4.3]에 대해 [그림 4.6]에서 예제 PPTL 파일을 가지고 설명할 것이다.

```
/countries/country/name
/countries/country/population[@year]
/countries/country/population_growth
/countries/country/infant_mortality
```

```
/countries/country/gdp_total
/countries/country/gdp_agri
/countries/country/gdp_ind
/countries/country/gdp_serv
/countries/country/inflation/year
/countries/country/inflation/factor
/countries/country/indep_date
 /countries/country/government
 /countries/country/ethnicgroups[@percentage]
 /countries/country/religions
 /countries/country/languages[@percentage]
 /countries/country/density[@xmlns:xlink]
 /countries/country/urban_pop[@xmlns:xlink][@xmlns:dbxlink]
 /countries/country/capital[@xmlns:xlink][@xmlns:dbxlink]
 /countries/country/cities[@xmlns:xlink][@xmlns:dbxlink]
 /countries/country/provinces
 /countries/country/encompassed[@xmlns:xlink][@xmlns:dbxlink]
 /countries/country/neighbor[@xmlns:xlink][@xmlns:dbxlink]
```

[그림 4.6] 예제 PPTL 파일

[그림 4.6]의 PPTL 파일은 총 22개의 PPTL을 가지고 있다. 이 때, 사용자가 질의문 “cities and inflation”에 대해 TMT를 50%로 설정하였을 때의 처리 과정을 설명해 보면 다음과 같다. 먼저 질의문에 쓰인 태그 이름 “cities”와 “inflation”에 대한 TMP를 구한 후, TMR을 구한다.

[그림 4.6] 파일에 대해 TMP(cities)[1]의 개수는 1이다. PPTL 파일의 경로 수는 22개이므로. TMR(cities) = $\frac{1}{22}\times 100(\%) = 4.5(\%)$ 로 얻어진다. [그림 4.6] 파일에 대해 TMP(inflation) [2] 의 개수는 2이다. TMR(inflation) = $\frac{2}{22}\times 100(\%) = 9.1(\%)$ 이다.

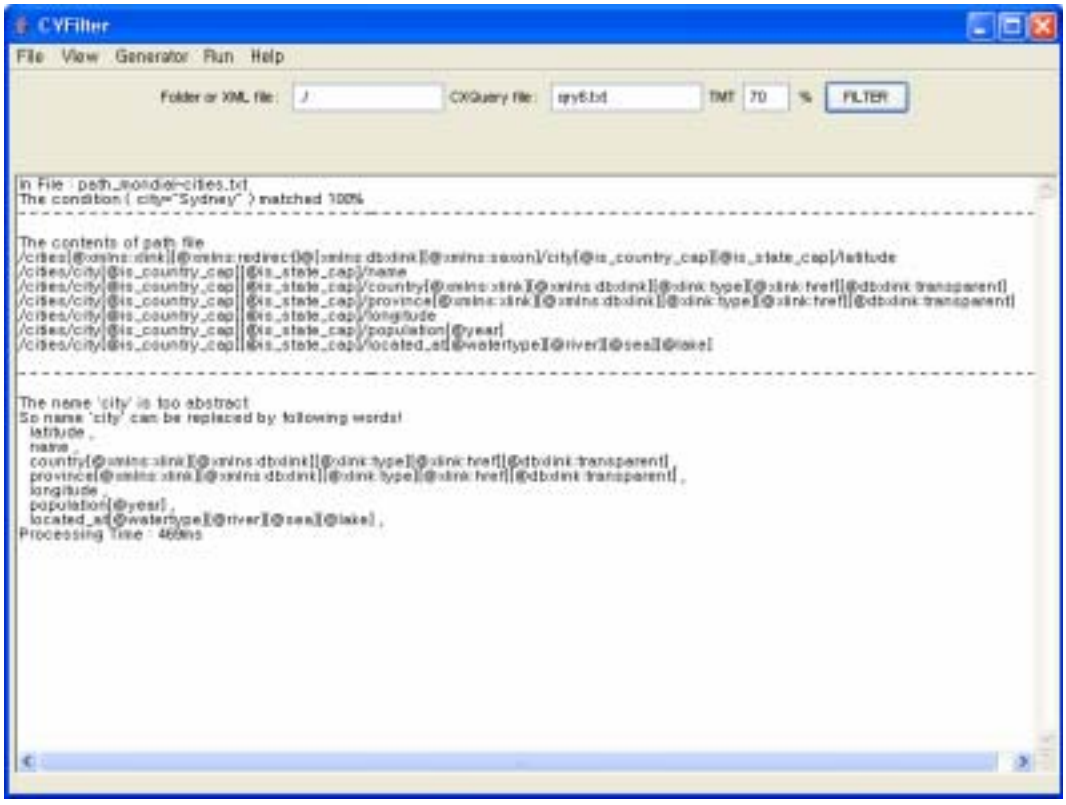
[1] TMP(cities)={/countries/country/cities[@xmlns:xlink][@xmlns:dbxlink]}

[2] TMP(inflation)={/countries/country/inflation/year, /countries/country/inflation/factor}

$50 > MAX(4.5,9.1) = 9.1$ 으로, $TMT > MAX(TMR(cities), TMR(\inf lation))$ 최대 TMR이 TMT를 넘지 않으므로 위의 질의문을 가지고 필터링을 수행한다.

이 값들을 정의함으로써 사용자가 보다 원하는 결과를 얻을 수 있는 태그 이름을 사용하도록 하고, XPath 질의 변환 시 다수의 변환된 XPath가 생성되지 않도록 하여 결과를 얻는 시간이 길게 걸리지 않도록 하였다. TMT를 넘는 태그이름이 CXquery 파일의 조건절에 사용되었을 때, 필터링을 수행하지 않는다. 사용자가 TMT를 지정하지 않으면 디폴트 값은 0이다.

[그림 4.7]은 사용자가 TMT를 70%로 설정한 경우이다. 사용자가 필터링을 수행할 때, TMT를 지정하게 되면 시스템은 조건절에 쓰인 각 태그들의 TMR을 구한다. 태그의 TMR 중 70%를 넘는 태그가 있으면 필터링을 수행하지 않는다.



[그림 4.7] TMT를 넘어선 데이터를 질의에 사용한 경우

CXquery 조건절에 쓰인 태그이름이 PPTL 파일에서 나타나는 TMT가 70%가 넘어서서 사용자에게 필터링을 수행할 수 없음을 알려주는 화면이다. 화면의 내용을 설명하면, 태그이름 “city”가 PPTL 파일 “path_mondial-cities.txt”에서 전체 경로 중 나타나는 횟수가 100%이다. 그리고, 경로 파일의 내용을 보여주고, 태그이름 “city” 아래에 있는 하위의 다른 엘리먼트, 애트리뷰트를 보여주면서 다른 태그이름을 사용하도록 메시지를 보여준다.

Ⅴ. 실험 및 분석

본 장에서는 사용자의 질의가 경로 기반의 질의를 사용하는 YFilter와 본 연구에서 제시한 경로 기반을 사용하지 않는 문서 구조에 독립적인 질의를 사용하는 CYFilter 시스템의 성능을 비교해 본다.

5.1 실험 환경 및 데이터

실험 환경은 윈도우는 Microsoft Windows XP 이고, CPU 는 Pentium 4, 1.90 GHz 램의 사양이다. 사용자가 질의를 던지고, 결과가 결과 파일에 다 저장될 때까지의 수행시간을 자바의 System 패키지의 currentTimeMillis 메소드를 사용하여 구하였다.

성능을 평가하기 위해 쓰인 스트리밍 XML 데이터로 XML 벤치마크용 XML 문서를 제공하고 있는 XMark 를 사용하였다. 여기에서 제공하는 XML 문서는 전형적인 전자 상거래인 온라인 옥션을 위한 것이다. 데이터는 크게 두 가지 타입의 종류이다. 하나는 물품 주문 배송 정보에 대한 명세를 나타내는 XML 문서이고, 다른 하나는 물품 주문자에 대한 명세를 나타내는 XML 문서이다. <표 5.1>은 두 종류의 XML 문서에 대한 명세표이다. 아래의 데이터를 가지고 데이터 수를 증가시키면서 시스템의 처리 시간을 측정해 본다.

<표 5.1> 실험 데이터 명세

	site*xx*.xml	people*xx*.xml
문서의 내용	주문 배송 정보를 담은 문서	주문자 정보를 담은 문서
한 문서 당 엘리먼트 개수	24 개	21 개
한 문서 당 애트리뷰트 개수	3 개	3 개
한 문서의 평균 크기	10kb	5kb
루트 엘리먼트	Site	site
최대 깊이	7	5

5.2 실험 평가

본 논문에서 CYFilter 시스템의 성능을 측정하기 위해 두 가지의 경우에 대해서 실험하였다.

- C1: 사용자가 문서의 구조를 정확히 알고 XPath 로 질의를 하는 경우
- C2: 문서의 경로 파일들이 생성되지 않은 상태에서 CXquery 을 사용하는 경우

C1 은 사용자가 XPath 질의어를 이용하여 질의를 하는 YFilter 을 이용하여, 스트림 XML 문서를 필터링을 한 경우이고, C2 는 CXquery 를 이용하여 질의를 한 경우(CYFilter 사용)이다. C2 의 경우는 CXquery 질의 파일을 사용하는 것으로, 사용자가 CXquery 로 질의하게 되면, 시스템은 CXquery 질의를 XPath 필터기의 입력에 맞는 XPath 형태의 질의 파일로 바꾸는 일을 한다.

위의 두 가지의 경우에 대해 데이터의 양이 증가될 때 그 추이에 대해서 실험을 하였다. 실험에 사용된 XPath, CXquery 질의문은 다음과 같다. <표 5.2>은 YFilter 에 쓰인 XPath 질의문이고, <표 5.3>은 YFilter 에 쓰인 CXquery 질의문과 변환된 XPath 질의문이다.

사용자가 “item”이 “item1”의 값을 가지는 XML 문서의 일부를 찾고자 하는 경우를 실험에 사용하였다. <표5.2>는 위의 질의를 가지고 각각의 DTD에 맞추어 XPath를 하나씩 생성한 것이고, <표 5.3>의 (a)는 CXquery이고, (b)는 CXquery 질의 처리기를 통해서 XPath로 변환된 질의문이다.

<표 5.2 > C1 (YFilter) 사용 질의문

XPath 질의문
XPath : /site/regions/africa/item[@id="item1"] /site/regions/asia/item[@id="item1"] /site/regions/namerica/item[@id="item1"] /site/regions/samerica/item[@id="item1"] /site/regions/europe/item[@id="item1"]

<표 5.3> C2 (CYFilter) 사용 질의문

(i) CXquery 질의문
CXQ1 : Item = “item1”
(ii) XPath 변환된 질의문
CXquery :

```
/site/regions/africa/item[text()="item1"]
/site/regions/africa/item//*[text()="item1"]
/site/regions/africa/item[@id="item1"]
/site/regions/africa/item/incategory[@category="item1"]

/site/regions/namerica/item[text()="item1"]
/site/regions/namerica/item//*[text()="item1"]
/site/regions/namerica/item[@id="item1"]

/site/regions/asia/item[text()="item1"]
/site/regions/asia/item//*[text()="item1"]
/site/regions/asia/item[@id="item1"]
/site/regions/asia/item/incategory[@category="item1"]
/site/regions/samerica/item[text()="item1"]
/site/regions/samerica/item//*[text()="item1"]
/site/regions/samerica/item[@id="item1"]
/site/regions/samerica/item/incategory[@category="item1"]

/site/regions/europe/item[text()="item1"]
/site/regions/europe/item//*[text()="item1"]
/site/regions/europe/item[@id="item1"]
/site/regions/europe/item/incategory[@category="item1"]
```

<표 5.4>는 이 실험에 쓰인 데이터에 대한 정보이다. Case1부터 Case4까지

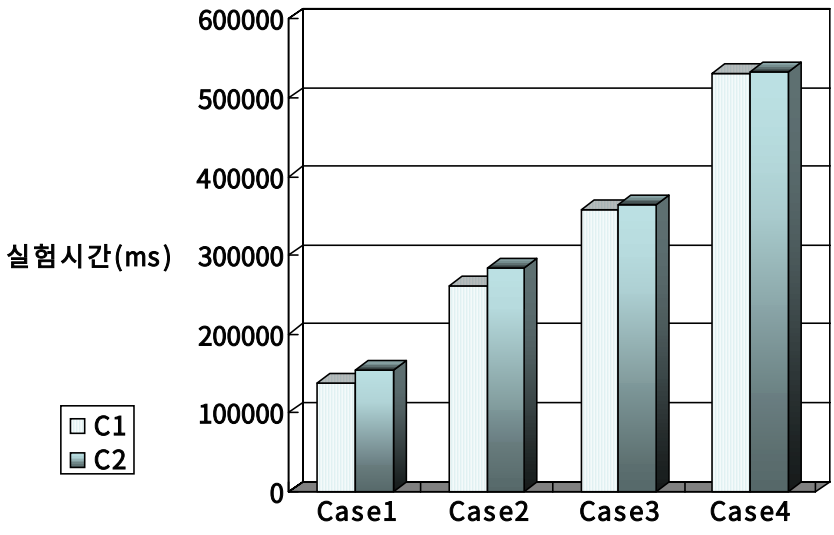
XML 문서의 개수를 증가시키고, 문서의 크기도 늘려서 실험을 하였다. 아래의 <표 5.5>는 <표 5.4>의 데이터를 가지고 실험한 결과이다.

<표 5.4> 실험 데이터

	Case1	Case2	Case3	Case4
문서 개수(개)	5000	10000	15000	20000
데이터크기(Mb)	41.0	64.9	72.8	97.7

<표 5.5> 실험결과

	Case1	Case2	Case3	Case4
C1 시간(ms)	137250	260487	357984	530859
C2 시간(ms)	158391	272766	363422	543719



[그림 5.1] 성능평가

위 성능평가에서 우리는 YFilter 와 CYFilter 가 질의문에 대한 결과를 보여주는 응답 시간에 큰 차이가 없음을 볼 수 있다. CXquery 질의를 사용한 경우(C2)가 XPath 질의를 사용한 경우(C1)보다 질의 처리 속도가 약간 느리다. 이것은 CYFilter 가 다량의 XML 문서를 읽으면서 그것들의 DTD 를 파악하고, 경로 파일을 생성하는 과정을 거친 후, XPath 질의로 만들어내는 차이가 있기 때문이다. 그러나 이 차이가 YFilter 와 비교하였을 때, 처리 속도에 크게 영향을 주지 않는다. 따라서 사용자는 XML 문서의 구조 독립적인 필터링을 지원하며 처리 속도와 정확도 면에서도 좋은 성능을 보이고 있는 CYFilter 를 이용할 것으로 기대된다.

Ⅵ. 결론 및 향후 연구과제

스트리밍 데이터는 빠르고, 언제 들어올지 예측할 수 없으며, 계속적으로 들어오는 특징을 가진다. 이러한 스트리밍 데이터가 XML 형태로 된 것을 스트리밍 XML 데이터라 한다. 웹 상에서 XML 은 문서 교환의 표준으로 많이 인정이 되고 있고, 활발히 문서 교환의 형식으로 사용된다. 이러한 스트리밍 XML 데이터에 대한 연구가 활발히 이루어지고 있다. 그러나 기존의 스트리밍 XML 데이터 필터링에 대한 연구들에서 사용되고 있는 질의 형식이 경로 표현식을 사용하고 있다. 이러한 질의 형식은 사용자가 문서의 구조를 알아야 하는 불편함을 초래한다.

본 논문에서는 XML 로 표현된 스트림 데이터에 대해 원하는 데이터를 검색할 때 문서 구조를 모르고서도 질의할 수 있고 원하는 결과를 얻을 수 있는 CYFilter 시스템을 개발하였다. 이를 위해 본 연구에서는 CXquery 개념을 도입, 질의 표현식을 사용하고 이를 XPath 로 변환함으로써 스트리밍 XML 데이터를 효과적으로 처리할 수 있도록 하였다. 스트리밍 XML 데이터 필터링은 사용자가 문서의 구조를 모르고도 검색하고자 하는 데이터의 이름과 값만을 주어 결과를 얻을 수 있으므로 문서의 구조를 모르는 사용자 및 XML 문서를 처음 다루어 경로 기반의 질의를 모르는 사용자에게도 도움이 될 것이다.

CYFilter 시스템은 XML 문서를 질의하는데 있어, 데이터가 있는 모든 경로를 사용하지 않고, 데이터 이름과 값을 사용하는 CXquery 를 사용하여, 필터링을 할 수 있도록 하였다. 이렇게 함으로써 사용자는 문서의 구조를 모르고도 원하는 XML 문서의 일부분을 얻어낼 수 있다. 비슷한 구조를 가진 XML 문서의 상이한 DTD 를 가질 경우, 사용자는 한 개의 단지 한 개의 질의문을 사용하면 된다. 기존의 시스템은 다른 DTD 에 대해서 각각 질의문을 생성하는 반면, CYFilter 은 한 질의문을 만들면 된다. 이 시스템은 사용자가 문서의 구조를 모르고도 질의할 수 있는

편리함을 제공할 뿐만 아니라 기존의 시스템과도 실험 결과 차이가 많이 나지 않는 등 거의 비슷한 수행 시간을 얻을 수 있었다.

태그 이름과 조건식의 비교값 사이에 다른 엘리먼트 혹은 애트리뷰트가 있는 경우를 처리하기 위해, 제시된 태그 이름 하위에 있는 엘리먼트나 애트리뷰트에 주어진 값을 대입시키는 방법을 사용하였다. 이 방식은 관계없는 태그 이름에 값을 대입시키는 문제를 초래한다. 이에 온톨로지의 개념을 도입하여 조건식의 비교값이 적절한 태그 이름에 대입될 수 있는 방안에 대해서 향후 과제로 제시한다.

참고 문헌

[1] W3C Consortium, XML 1.0 (Second Edition), W3C Recommendation 06 Oct. 2000, available at http://www.w3.org/TR/REC-xml.

[2] S. Babu and J. Widom, Continuous Queries over data Streams, In Proceeding of the SIGMOD, pp.109-120, 2001.

[3] J. Chen, D. J. Dewitt, F. Tian and Y. Wang, NiagaraCQ:A Scalable Continuous Query System for Internet databases, In Proceeding of the SIGMOD, pp.379-390, 2000.

[4] M. Altinel and M. J. Franklin, Efficient Filtering of XML Documents for Selective Dissemination of Information, In Proceedings of the 26th VLDB Conference, pp.53-64, 2000.

[5] S.Bong, D. Chamberlin, M.F. Fernanderz, D. Florescu, J. Robie, and J. Simeon, XQuery 1.0 : An XML Query Language, W3C Working Draft Nov 12, 2003, available at http://www.w3.org/TR/xquery/.

[6] W3C Consortium, XML Path Language(XPath), version 2.0, W3C Recommendation Nov 12, 2003, available at http://www.w3.org/TR/xpath20.html.

[7] 이월영. “XML 데이터베이스에서 문서 구조 독립적인 질의 처리 기법”. 이화여자대학교 과학기술대학원 박사학위청구논문. 2004.

[8] W. Lee, H. Yong, A Query Expression and Processing Technique for an XML Search Engine,

LNAI Vol. 3488 Foundations of Intelligent Systems, ISMIS 2005: 15th International Symposium on Methodologies for Intelligent Systems, Saratoga Springs, NY, USA, pp.266-275, May 2005.

[9] Y. Diao, H. Zhang and M. J. Franklin, YFilter : Efficient and Scalable Filtering of XML Documents, ICDE, 2002.

[10] Y. Diao and M. J. Franklin, High-Performance XML Filtering: An Overview of YFilter, IEEE Computer Society Technical Committee on Data Engineering, 2003.

[11] Y. Diao, H. Zhang and M. J.Franklin, NFA-based Filtering for Efficient and Scalable XML Routing, Technical Report, USB/CSD-1-1159, 2001.

[12] T.J. Green, G. Miklau and D. Suciu, Processing XML streams with deterministic automata, In Proceedings of IEEE Conference on Database Theory, 2003.

[13] Berkeley University, Yfilter, available at http://yfilter.cs.berkeley.edu/code_release.htm

[14] Sun, Java API docs, available at http://java.sun.com/j2se/1.5.0/docs/api/index.html.

[15] 송민영, 시각적 XML 질의어 자동 생성을 위한 사용자 인터페이스 구현, 이화여자 대학교 과학기술대학원 석사학위청구논문, 2002.

ABSTRACT

Streaming XML Data Filtering using CXquery

`

Depart of Computer Science and Engineering

Ewha Institute of Science and Technology

Ewha Womans University

Kim, So Ra

In distributed computing environments, such as Web Services, data and application integration, and personalized content delivery, XML(Extensible markup Language) is used as a standard format to exchange documents over the Internet. Therefore, there have been many proposals for streaming XML data filtering.

Queries in these systems are expressed in a XML query language such as XPath and XQuery. These queries adopted the navigational content-based queries and users have to write the paths which contain data name for querying. When we use these queries, we should know the structures of XML documents. If users don't know the structures, it is very inconvenient for them to search useful results. In order to relive users' inconveniences, CXquery was proposed as a XML query language. Contrary to the traditional query languages, query expressions in CXquery consists of only data names and data values. CXquery enables users to search XML documents regardless of their structures.

And we proposed CYFilter(Chamois YFilter). It is a streaming XML data

filtering system using CXquery. It consists of 3 modules such as Path Generator, CXquery processor and YFilter(XPath-based filtering system) to handle CXquery files.

Path Generator makes PPTP file(Posible Path To Leaf file) of DTD documents and it provides the needed paths for CXquery file to be transformed into the XPath query file. CXquery processor transforms CXquery file into XPath query file. This transformation is needed because YFilter is a XPath-based filterinig system. YFilter runs filtering using streaming XML documents and XPath query file. The results matched to the queries are saved into text file.

CYFilter is implemented with Java and can function as streaming XML data filtering system. Furthermore, it can provide users to edit documents such as XML and DTD. Users can run filtering regardless of structures of documents and it gives for users convenient.

We conducted diverse experiments to show that CYFilter achieves as a good performance as existing filtering system.