XML을 이용한 구조적 문서 생성 및 탐색을 위한 깊이중심분할 색인기법에 관한 연구

양 옥 렬† · 이 용 주††

요 약

본 논문은 정보검색을 위한 용어들 간에 존재하는 관련정보인 시소러스를 이용하여 정보 검색 시스템의 검색 성능을 향상시키기 위한 구조적 문서를 생성하고 이를 검색하는 검색 기법에 대하여 연구하였다. 이를 위해 시소러스를 이용한 구조적 문서의 탐색을 위한 깊이중심분할 색인(DODI : Depth-Oriented Decomposition Index) 기법을 제안하였으며, 또한 시소러스를 이용한 색인 기법으로서 효과적인 정보 검색이 가능하도록 검색 알고리즘을 통해 연관관계의 정보들에 대한 검색이 가능하도록 하였다. 또한 색인기법에 의해 생성된 구조적 문서는 OpenXML을 통해 데이터베이스 내에 저장되고, ForXML 메소드를 이용하여 재구성된 XML 문서를 생성하도록 구조적 문서 저장 시스템을 구현하였다.

A Study on the Depth-Oriented Decomposition Indexing Method for Creating and Searching Structured Documents Based-on XML

Ok-Yul Yang† · Yong-Ju Lee††

ABSTRACT

The goal of this study is to generate a structured document which improves the performance of an information retrieval system by using thesaurus, information on relations between words (terms), and to study on the technique for searching this structured document. In order to accomplish this goal, we propose a **DODI** (Depth-Oriented Decomposition Index) technique for the structured document and an algorithm to search for related information efficiently through this index technique that uses a thesaurus. We establish a storage system by which the structured document generated by this index technique is saved in a database through OpenXML and XML documents are generated through ForXML methods.

키워드 : 시소러스(Thessurus), 구조적 문서(Structured Documents), 색인(Index), 검색(Search), 탐색(Retrieval), 엘스엠엘(XML)

1. 서 론

정보통신산업의 급속한 발전과 함께 다가온 21세기의 정보 서비스 분야는 컴퓨터 산업뿐만 아니라 모든 산업에 걸쳐 효율적인 정보 검색을 필요로 하고 있다. 이제 정보 검색 서비스는 정보 컨텐츠의 다양화, 정보의 분산화, 대용량화, 멀티미디어화 등의 광범위하면서도 세부적인 형태로 정보를 다루게 되었다. 인터넷의 확산과 함께 정보의 양이 증가하고 저장된 도메인 데이터베이스에 대한 사용자 요구가 다양해지면서 보다 의미적으로 일치하면서 정밀한 검색을 할 수 있는 정보 검색 방법의 개발이 요구되고 있다[1]. 정보검색시스템은 사용자의 질의요청에 대하여 대량의 정보를 저장하고 있는 데이터베이스로부터 효율적인 검색을 통해 결과를 추출하는데 그 목적이 있는데, 현재 사용되는 대부분의 정보검색시스템의 경우, 사용자 질의에 대한 질의 결과를 얻는 방법으로 검색 키워드에 대하여 관련 문서 데이터를 결과로 출력한다[2, 3]. 이러한 검색기법은 검색을 위해 사용되는 색인어들의 집합을 구성하고 이에 따라 사용자가 검색어 사이에 부울연산을 통해 질의 결과를 얻는다[4]. 그러나, 검색시스템으로부터 사용자 질의를 통해 얻어진 문서의 색인은 완벽하게 사용자의 질의어에 포함된 키값과 일치하지 않는데, 이러한 문제를 해결할 수 있는 가장 적합한 방법이 데이터베이스화 된 문서에 대하여 관련어의 관계를 표현한 시소러스 이용한 검색기법을 이용하는 방법이다[5].

정보 검색 과정에서 질의어에 표현되지 않은 용어들은 특정문서에 대한 키워드가 색인되었을 경우, 문서가 사용자

※ 이 논문은 2001년도 원광대학교의 교비 지원에 의해서 수행됨.
† 정 회 원 : 휴먼미디어테크 연구소장
†† 정 회 원 : 원광대학교 컴퓨터 및 정보통신 공학부 교수
논문접수 : 2002년 9월 30일, 심사완료 : 2002년 12월 5일

요구사항과 일치하더라도 결과로 검색된 정보에는 나타나지 않을 수 있다. 이러한 문제를 해결하기 위해 서버에 저장된 데이터베이스 정보에 의미관계를 표현하는 시소러스 검색 과정이 이용되는 것이다[6].

일반적으로 시소러스의 구축은 전문가에 의한 수작업 방식의 구축방법과 자동으로 구축되는 방식의 두 가지 방식으로 나누며, 검색의 효율을 높이기 위해 질의 처리나 색인 과정에서 두 가지 구축방법을 이용한다[7].

자동으로 구축되는 시소러스는 동시 출현 빈도수가 높은 색인어 정보데이터나 통계정보 계산을 위해 매우 큰 구축 비용을 요구한다. 또한 이는 개념들 사이의 연관정보만을 표현함으로써 충분한 데이터베이스 내의 정보를 표현하지 못하는 단점을 가지고 있다[7, 8].

전문가에 의한 시소러스 구축방법은 시소러스 개념들 사이의 의미구조를 명확하게 반영할 수 있다는 장점을 갖는다. 즉, 용어들 사이에 존재하는 최상위어(TT), 상위어(BT), 하위어(NT), 관련어(RT) 등의 다양한 관계를 표현하는데 적합하다. 그러나 모든 관계를 전문가가 직접 명시해야 하므로 구축비용이 많이 들며 일관성과 객관성을 유지하는데 어려움이 있다[7, 9-11]. 그러나 대부분의 경우 시소러스는 도메인 전문가에 의해 해당되는 구축 데이터베이스 내에 저장된 문서 정보의 특징에 따른 수동 색인 방법이 일반적이라고 할수 있다. 이는 정보 검색 시스템의 성능이 저장된 문서의 특징과 도메인 전문가에 의해 결정되는 경우가 크기 때문에 그만큼 사용자 질의가 저장문서의 형식과 얼마나 근접해 있는가에 따라 매우 의존적이라는 의미이며, 또한 구축되는 시소러스의 정보 형태에 따라 크게 달라질 수 있기 때문이다.

도메인에 대한 정확한 정보를 사용자가 요구하는 정확한 질의에 대한 결과를 구성하기 위해서는 도메인 전문가는 초기 질의에 대한 피드백을 통해 도메인 정보에 대한 반복적인 질의 재형성 과정을 밟게 된다. 이러한 질의 재형성을 위해서는 크게 수동 질의 재형성 방법과 자동 질의 재형성 방법이 사용된다.

본 논문에서 제안한 색인 모델은 객체지향적인 기법을 이용하여 데이터베이스에 대한 시소러스를 반자동으로 구축하고 유지하며 일반적인 색인기법에서 지원하지 못하는 계층적 트리구조에서 각 레벨을 기준으로 하는 검색을 통해 색인 검색에 대한 제한이 사용자별 서비스 범위에 따라 제어할 수 있다. 이는 세부적으로 계층을 형성하는 더미노드(dummy node)를 사용하여 각 계층에 대한 동일한 엘리먼트 형식을 갖도록 유도하고 실제 검색에서는 각 엘리먼트의 상·하위 관계에 대한 전이적 성질로 정의되는 객체의 계층적 구조에 파생하는 상속성과 후보객체들 간의 묵시적인 집성화 및 연관화 관계를 갖도록 하였다. 이 방법은 구축 전문가가 객체들 사이의 관계를 개별적으로 모두 명시하지 않으므로 구축비용을 절감할 수 있다. 즉, 본 논문에서 제안한 시소러스를 이용한 깊이중심분할 색인은 사용자가 보다 데이터베이스 시스템에서 정보들을 체계적이고 의미적으로 적절한 내용을 신속하게 저장 검색할 수 있도록 하게 되며, 도메인에 대한 정확한 지식이 없는 일반 사용자나 시스템 자체에 대한 정확한 지식이 없는 사용자에게도 각 계층별 구조검색과 동의어 검색을 통해 필요한 정보를 검색할 수 있도록 하게 한다. 또한 새로운 시소러스 색인에 대한 검색기법 알고리즘을 통해 시소러스 구축시에 발생하는 여러 문제점들을 해결하고 검색 성능을 높여 보다 신속한 구조적 문서의 저장, 생성 그리고 검색을 하는데 그 목적이 있다.

본 논문은 기존의 시소러스 색인 기법이 가지고 있는 문제들을 해결하기 위해 새로운 방법의 색인기법을 제안하고, 이렇게 제안된 시소러스 색인 모델은 구조적 형식을 갖는 문서에 적용함으로써 구조적 문서의 검색과 제어를 효율적으로 유지하는데 연구의 목적을 가지고 있다. 이를 위해 관계형 데이터베이스 저장 시스템을 이용한 구조검색이 가능하도록 엘리먼트, 엔티티, 속성 등의 각 노드별 정보의 추출이 가능하도록 하는 구조적 문서 단위의 시소러스 검색을 제안하였고, 저장과 색인을 위한 구조적 문서를 XML을 통해 생성하고 생성된 문서의 구조적 트리 생성을 위해 DOM 인터페이스를 활용하여 구현하였다. 탐색기법은 이러한 DOM의 계층적 구조를 활용한 계층단위의 색인 방법을 의미한다. 또한 문서의 효율적인 검색과 저장을 위해 구조적 문서를 문서 구조의 의미적 연관관계에 따라 분할하여 저장하도록 설계하였다. 그러나, 현재 XML 문서를 효율적으로 사용하기 위해서는 반드시 필요한 요소로는 XML 지원 웹 브라우저와 XML 문서를 쉽고 유용한 정보로 만들기 위한 검색 및 저장 시스템의 보편화가 필수적이다. 그러나 현재 문서 검색 시스템의 대부분은 내용정보와 문서의 구조정보를 동시에 효과적으로 검색할 수 있는 형태를 지니고 있지 못하다[12].

본 논문에서는 이러한 용어들 간에 존재하는 관련정보를 이용하여 정보 검색 시스템의 검색 성능을 향상시키기 위한 구조적 정보를 구축하고 유지시키도록 시스템을 설계하였다. 이러한 요구사항을 효과적으로 반영하기 위해 시소러스를 이용한 구조적 문서의 탐색을 위한 깊이중심 분할 색인 모델을 제안하였으며 이러한 시소러스를 이용하여 효과적으로 검색 할 수 있도록 검색 알고리즘 제안을 통해 연관관계의 정보들에 대한 검색이 가능하도록 하였다. 또한 색인기법과 구조적 문서를 통해 얻어진 DOM 트리 정보를 이용한 구조적 문서 저장 시스템을 구현하였다.

본 논문의 구성은 2장에서는 깊이중심분할 색인 기법인 DODI의 색인기법에 대해 기술하고, 3장에서는 제안한 DODI를 기반으로 하는 검색 알고리즘에 대하여 기술하고, 4장에

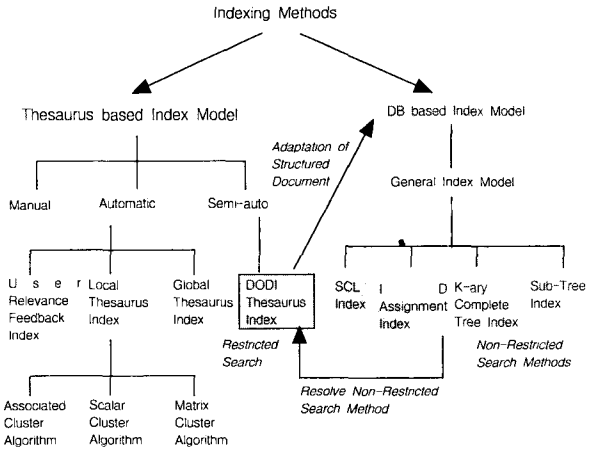
서는 제안한 DODI 색인 기법에 대한 한의학 문헌정보를 대상으로 색인 및 검색기법에 대한 구현 및 성능평가를 기술하고, 마지막으로 5장에서 결론을 맺는다.

2. 깊이중심분할 색인 기법

2.1 기존 색인 기법의 문제점

일반적으로 데이터베이스 색인을 위한 색인 기법으로는 단순 일치 리스트, ID 할당, K-ary 완전트리, 서브트리 색인 기법 등이 있다.

이러한 색인 기법은 데이터베이스를 기반으로 하여 사용되고 있는데 앞서 언급한바와 같이 깊이를 중심으로 하는 제한적 검색을 사용하기에 제약이 많다. 본 논문에서 제안한 색인 기법은 시소러스를 기반으로 하는 색인 기법 중에 반자동 형식의 색인 기법으로 데이터베이스 기반 색인 모델의 단점을 보완하고 DB내에서 구조적 문서 처리가 가능하도록 설계하고자 한다. (그림 1)은 색인기법에 대한 분류이다.



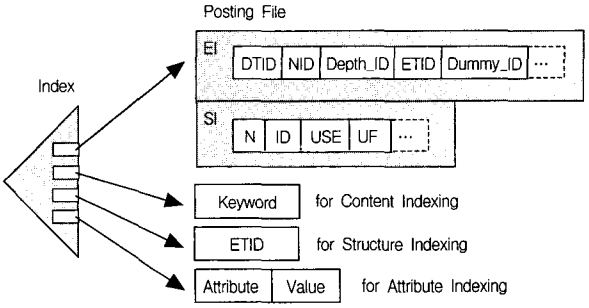
(그림 1) 색인 기법의 분류

2.2 시소러스를 이용한 DODI 색인 기법 제안

논문에서 제안하는 색인 기법은 기존의 단순 일치 리스트 색인, K-ary 완전트리 색인, ID 할당 색인 및 서브트리 색인 기법 등이 제공하지 못하는 특정 범위나 사용자에 대한 제한적 검색이 가능한 구조적 문서의 특성을 고려하여 각 계층별로 존재하는 엘리먼트의 형식을 동일한 레벨에 존재하도록 배치하여 구조적 문서의 제한 검색이 가능하도록 시소러스를 이용한 깊이중심분할 색인(DODI : Depth Oriented Decomposition Indexing) 모델을 설계하였다. 이를 위해 단말 노드에 해당하는 인스턴스에 대한 노드 레벨을 맞추는데 더미노드(dummy node)를 사용하였다. 더미노드는 계층별 제한 검색에서 엘리먼트의 형식을 동일한 레벨에 존재하도록 하기 위해 추가하는 가상의 정보 노드이다. 이를 통해 DODI 모델링은 특정 조건에 대하여 기준 노드를 정하고 정해진 노드로부터 특정 노드에 대한 계층정보와 순서정보를 간단한 문자열 조작으로 가능하고 유일한 노드의 할당 ID값을 이용하여 직접 노드 검색이 가능하므로 제한적인 검색이 가능하다.

2.2.1 색인구성

구조적 문서의 특성을 고려하여 문서의 다양한 질의를 효율적으로 처리하기 위해 색인 구조를 설계하는데, 이러한 색인은 구조적 문서에서 추출된 구조정보를 이용하여 생성된 정보에 대하여 크게 내용 색인, 구조 색인, 속성 색인의 색인으로 구성된다. 본 논문에서는 이들 3가지 색인을 하나의 포스팅을 통해 구성하였다. 그 구성은 아래 (그림 2)와 같다.



(그림 2) 색인 구조

2.2.1.1 내용 색인

내용 색인(content index)은 구조적 문서에 존재하는 실제 인스턴스를 검색하는데 필요한 색인으로 구조적 문서에서 추출된 색인어로 구성된 색인파일과 색인어가 포함된 문서와 문서내의 엘리먼트 정보를 나타내는 포스팅으로 구성된다. 포스팅에 사용되는 필드 정보는 *DTID(Document Type ID)*, *NID(Node ID)*, *Depth_ID(Node Level Depth)*, *ETID(Element Type ID)*, *Dummy_ID(Dummy Node ID)* 등 5가지로 구성된다. 이들 필드 정보는 구조 색인이나 속성 색인에서도 동일하게 사용 가능하다.

2.2.1.2 구조 색인

구조 색인(structure index)은 구조검색을 지원하기 위한 색인으로서 문서의 논리적인 구조 및 계층적인 구조를 정보 손실 없이 표현하는데, 이를 통해 현재 엘리먼트 노드를 중심으로 부모, 자식, 조상, 자손, 형제 노드를 검색할 수 있다. 이는 동일 문서 내에 존재하는 엘리먼트간의 순서 정보도 표현 가능함을 의미한다.

2.2.1.3 속성 색인

속성 색인(attribute index)은 엘리먼트에 존재하는 엘리먼트 속성정보를 검색하는데 사용되는 색인이다. XML 형

식의 구조적 문서를 DOM을 이용하여 구성된 모든 트리의 노드를 객체로 구성하고 이러한 객체로 존재하는 엘리먼트 이름, 엘리먼트 인스턴스, 속성 이름, 속성 인스턴스 등의 정보 중에 속성의 이름과 인스턴스 값 만을 추출해 포스팅한다. 제안한 색인에서는 속성 색인을 통해 가상으로 노드 정보인 더미노드에 대한 색인을 속성색인 기법을 이용하여 색인 하였다.

2.2.2 구조적 DODI 표현 구조

구조적 문서의 문서트리에 더미 노드를 추가함으로써 동일한 인스턴스 노드 레벨의 구성이 완성되면 효율적인 노드 관리와 검색을 위해서 각 엘리먼트 노드에 대한 색인 작업을 수행한다.

본 논문에서 제안한 DODI 색인 기법은 다음과 같은 기능을 포함하도록 제안하였다.

① 구조화된 문서 구조의 깊이를 중심으로 수평적/수직적 우선검색의 형식을 갖는 제한 검색 기능
② 구조화된 문서를 기능별 분할 저장 및 색인 기능
③ 문서 정보, 엘리먼트 정보, 깊이 정보, 더미노드 정보의 표현 기능
④ 동의어 정보에 대한 색인 기능
⑤ 구조적 관계에 대한 표현 기능
⑥ 속성정보를 이용한 서브트리 추출을 통한 구조적 문서 생성 기능
⑦ 인스턴스 표현 계층인 단말노드의 특성을 레벨 깊이를 통해 검색하는 기능
⑧ 동일 계층에 존재하는 구조적 문서 트리의 속성 정보 보장 기능

이를 위해 논문에서는 노드들에 대한 색인을 크게 엘리먼트 정보 색인(*EI* : *Element Information Index*)과 동의어 정보 색인(*SI* : *Synonym Information Index*)으로 구성하여 문서정보 및 엘리먼트 정보, 엘리먼트 깊이 정보, 더미노드 정보를 표현할 수 있게 설계하였다. 엘리먼트 정보색인(***EI***)은 노드에 대한 데이터 타입과 노드 ID, 깊이 정보, 엘리먼트 타입, 더미노드의 유무를 판별할 수 있는 플래그 정보로 구성하였다. 동의어 정보색인(***SI***)을 위해서는 노드정보와 노드의 고유한 계층정보 ID, 동의어에 대한 우선어(USE)와 비우선어(UF) 관계 정보를 갖도록 설계하였다.

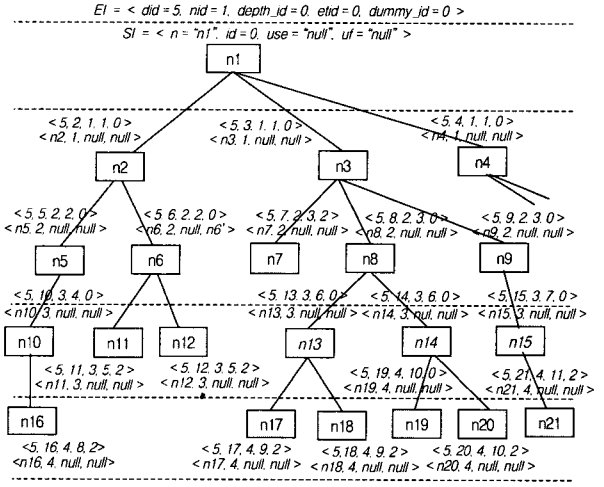
$$EI = < DTID,\ NID,\ Depth\_ID,\ ETID,\ Dummy\_ID > \quad ①$$

- *DTID* : *DocumentTypeID*
- *NID* : *NodeID*
- *Depth_ID* : *LevelDepthID*
- *ETID* : *ElementTypeID*
- *Dummy_ID* : *DummyNodeID*

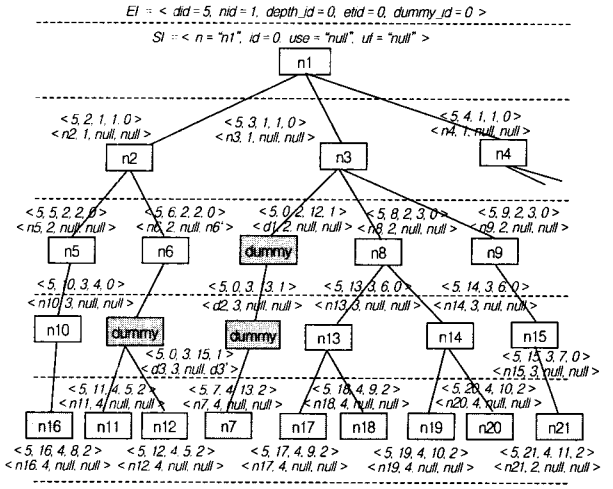
$$SI = < N,\ ID,\ USE,\ UF > \quad ②$$

- *N* : *NodeName / DummyNodeName*
- *ID* : *HierarchicalID*
- *USE* : *Use*
- *UF* : *Use For*

아래 그림은 레벨 깊이를 수정한 구조적 문서 트리구조에 엘리먼트 정보색인(***EI***)과 동의어 정보색인(***SI***)을 할당하여 구조적 문서 트리의 각 노드에 색인 정보를 추가한 상태이다. 추가된 색인 정보를 통해 각 노드의 레벨의 위치와 깊이를 쉽게 검색 할 수 있다. 그러나 이는 서로 다른 레벨의 노드가 같은 레벨에 존재함으로 인해 구조적 문서에서 레벨에 의한 제한적 검색이 어렵다는 단점을 가진다. 따라서 (그림 3)의 색인에 대하여 더미노드를 추가하여 색인 한 구조적 문서가 아래의 (그림 4)이다.



(그림 3) EI/SI를 할당한 구조적 문서 트리구조



(그림 4) EI/SI 및 더미노드를 추가한 구조적 문서 트리구조

(그림 3)은 ***EI***와 ***SI*** 색인 정보를 추가한 문서 트리구조이다. 각 노드는 ***EI***와 ***SI*** 정보를 갖게되는데, 아래의 예는

문서번호가 5이고 트리구조의 최대 깊이가 4인 레벨 깊이를 갖는다고 할 때, 노드 중 ***n6***의 ***EI***와 ***SI***를 살펴보면 식 (2.1)과 같다.

$$EI_{n6} = < dtid = 5,\ nid = 6,\ depth\_id = 2,\ etid = 2,\ dummy\_id = 0 > \quad (2.1)$$

여기서 ***nid***는 노드의 고유한 ID를 의미하는데, 만일 노드가 일반적인 노드정보가 아닌 가상의 노드 정보인 더미노드일 경우 ***nid***의 값을 0으로 세팅한다. 깊이정보인 ***depth_id***는 최상위 레벨을 0으로 시작하며 ***n6***의 깊이정보 값으로 2를 갖는다. 각 노드에 해당하는 엘리먼트의 타입 정보를 저장하기 위해 사용되는 ***etid***는 상위 속성 정보 값에 대하여 하위 노드로 존재할 때, 같은 부모노드를 갖는 자식노드들에 대하여 동일한 엘리먼트 타입 정보(***etid***)값을 갖도록 한다. 부모와 자식간의 엘리먼트 타입 정보는 1씩 증가하도록 설계하였다. 따라서 노드 ***n6***의 부모노드인 ***n2***의 엘리먼트 정보는 $EI = <5, 2, 1, 1, 0>$이므로 ***n6***는 $etid = 2$의 값을 가지며 ***n6***와 동일한 부모인 ***n2***를 갖는 ***n5***와 ***n7***도 $etid=2$라는 엘리먼트 속성 값을 갖는다. 또한 현재 노드에 대한 더미노드 유무를 나타내는 플래그 정보 값인 ***dummy_id***의 값은 더미노드일 경우 1로 일반적인 노드일 경우 0의 값을 갖게 되는데 ***n6***는 일반노드에 해당하므로 0으로 세팅한다.

n6의 ***SI*** 값은 해당 노드가 갖게되는 동의어에 관련된 정보를 갖게되는데 이는 다음 식 (2.2)와 같은 값을 갖게된다.

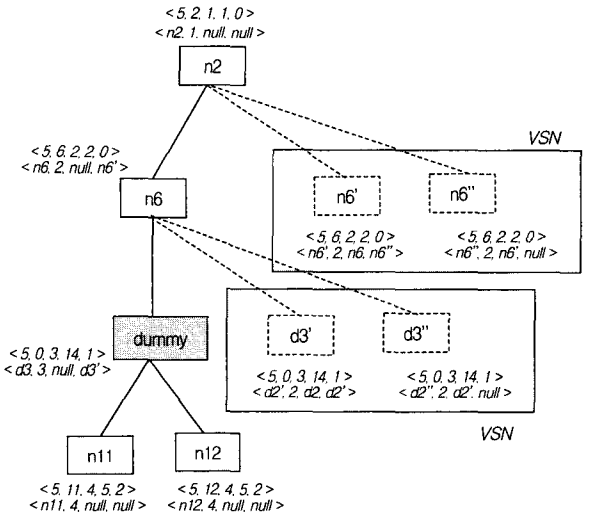
$$SI_{n6} = < n = \text{"n6"},\ id = 2,\ use = \text{"null"},\ uf = \text{"n6'"} > \quad (2.2)$$

만일 해당 노드에 대한 동의어가 존재할 경우 우선어 ***use***와 비우선어 정보 ***uf***를 갖게 된다. 구조적 문서의 트리에 존재하는 노드의 경우 우선어 정보는 ***null***을 갖도록 설계하였으며 해당되는 동의어 정보를 비우선어로 하여 ***uf*** 값을 설정하도록 구성하였다. 위의 그림에서는 예를 들어, ***n6*** 노드에 대하여 ***n6'***이라는 동의어 정보가 있다면 이에 대하여 비우선어 정보 ***uf*** 에 해당 인덱스 값을 갖도록 하였다.

더미 노드의 경우 ***EI***와 ***SI***의 값에 대하여 특정 값을 할당한다. 예를 들어, ***n6***의 자식노드로 존재하는 더미노드의 ***EI***값과 ***SI***값을 살펴보면 엘리먼트 정보는 $EI_{d3} = <5, 0, 3, 14, 1>$의 값을 갖는다. 여기서 ***nid***의 값을 0으로 세팅하는 것은 현재 노드가 더미노드임을 의미하는 값이다. 또한 해당 더미노드는 ***dummy_id***값을 1로 세팅한다. ***dummy_id***의 값을 1로 세팅하는 것 이외에 ***nid***의 값을 0을 할당하는 방식으로 중복된 더미노드에 대한 특성정보의 표현을 하는 이유는 ***nid***값 자체만을 이용하여 노드를 검색할 때, 더미노드 상위노드와 더미노드 하위노드간에 존재하는 전이적 성질에 대한 검색에서 사용되기 위함이다. 그밖에 깊이정보는 ***n6***의 하위 노드임을 의미하도록 1이 증가된 3을 가지며 엘리먼트 속성은 별도의 속성 정보 값을 할당한다. 또한 ***SI***의 값은 $SI_{d3} = < d3, 3, null, d3' >$의 값을 갖는다. ***d3***는 더미노드의 ID값이고 노드 속성 값으로 3을 갖는다. 이는 일반적인 노드에서 상위의 부모 속성 값에서 1이 증가된 값이다. 또한 ***d3***가 동의어를 가지고 있을 때 ***uf***의 값을 갖는데, 위의 예에서는 ***d3'***이라는 비우선어를 포함하고 있음을 보여준다. 더미노드에서도 역시 우선어인 ***use***값은 구조적 문서 트리에서 존재하는 노드일 경우 ***use***값을 ***null***로 갖도록 설계하였다.

2.2.3 동의어 정보 색인을 이용한 가상 동의어 노드의 표현

동의어 정보 색인을 위한 ***SI***값은 구조적 문서 트리에 존재하지 않지만 시소러스 검색 과정에서 일반 사용자의 질의에 대한 검색에서 사용자가 입력하는 키워드에 대한 동의어 검색에 사용한다. 이를 위해 별도의 동의어 정보 색인을 저장하고 검색 질의가 발생하면 SI 노드 색인 정보 인덱스에서 동일한 키워드를 검색하고 검색된 키워드에 동의어(우선어)로 존재하는 실제 노드를 검색하도록 설계하였다. 이때 생성되는 동의어들은 실제 노드에 존재하지 않는 노드들로써 이를 가상 동의어 노드(***VSN : Virtual Synonym Node***)라 한다. 이러한 가상 동의어 노드는 ***SI***를 이용해 표현할 수 있는데 아래 그림과 같이 ***n6***와 더미노드 ***d3***에 대한 ***VSN***을 표현할 수 있다.



(그림 5) SI를 이용한 가상 동의어 노드의 표현

이때 가상 동의어 노드의 조건은 아래와 식 (2.3)과 같이 표현할 수 있다.

$$SI_{VSN} = < n, id, use, uf > \text{단, } use \neq null \quad (2.3)$$

즉, 우선어인 ***use***의 값이 null이 아닌 ***SI***가 색인 정보 중에 가상 동의어 노드임을 알 수 있게 한다. 이를 통해 실제 동의어에 의한 검색에서는 대표 ***SI*** 노드를 통해 계층관계

및 연관관계 정보 노드를 검색할 수 있다. 대표 ***SI*** 노드란 ***n6***과 ***d3***과 같은 트리에 실제로 존재하는 노드들을 의미하는데, 이들의 특징은 ***SI*** 정보 내에 ***use***값을 항상 null로 갖게된다. 또한 ***uf***에 값이 반드시 null이 아닌 값으로 세팅되어 있는 ***SI***값을 가진다. 이때 위의 그림에서 ***n6'***와 ***n6"***는 모두 ***n6***노드를 우선어로 갖는 동의어들이다. ***n6'***는 우선어로 ***n6***을 가지며 비우선어로 ***n6"***를 ***uf***값으로 설정한다. 또한 비우선어의 마지막에 위치한 ***n6"***는 우선어로 ***n6'***를 가지며 ***n6"***에 대한 비우선어 ***uf***를 null로 세팅하여 동의어의 마지막임을 알 수 있다. 만일 추가로 동의어를 입력할 때에는 ***n6"***의 비우선어에 추가될 동의어를 설정하면 동의어 추가가 가능하다. 더미노드인 ***d3***의 경우도 ***d3'***과 ***d3"***을 동일한 규칙의 동의어 관계를 나타낸다.

2.2.4 더미노드의 전이적 성질 표현

본 논문에서 제안한 더미노드는 한 레벨에 존재하는 다양한 타입의 엘리먼트 노드에 대한 계층적 트리구조의 정렬을 위해 사용하였다.

아래 그림은 더미노드를 이용하여 발생하는 노드의 변화에 대하여 원래의 구조적 문서 트리구조를 복원할 수 있도록 취해야하는 더미노드의 상위노드와 하위노드에 대한 전이적 성질에 대한 표현이다. 우선 노드 n_1에 대하여 ***EI*** 값이 $EI_{n1} = <dtid_i, n_i, e_i, depth_id_i, dummy_id_{i-0}>$이고 ***SI***값이 $SI_{n1} = <n_1, id_k, use, uf>$이라고 하고 이 노드의 하위 노드에 더미노드가 존재한다면 더미노드의 ***EI***값과 ***SI***값은 다음과 같다. $EI_{d1} = <dtid_i, 0, e_{i+1}, depth\_id_{i+1}, dummy\_id_{i-1}>$, $SI_{d1} = <d_1, id_k, use, uf>$이다. 이때 이들의 관계는 부자관계를 이루는데 이러한 n_1는 d_1를 하부 노드로 가지고있는 연관 관계($sub\_con\text{-}of/W_{n1,d1}$)를 갖게 된다. 또한 d_1는 n_3를 역시 하부노드로 갖는 연관관계($sub\_con\text{-}of/W_{d1,n3}$)를 갖게 되는데 이에 대한 수식은 다음과 같다.

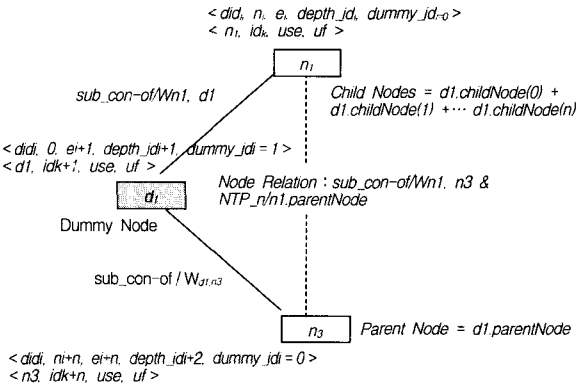
더미노드에 대한 상위 노드는 하나이지만 더미노드에 대한 하위 노드는 여러 개 일수 있는데, 그 중 n_3가 더미노드 d_1의 하위 노드라고 한다면 이에 대한 표현은 $EI_{n3} = <did_i, n_{i+n}, e_{i+n}, depth\_id_{i+2}, dummy\_id_{i-0}>$, $SI_{n3} = <n_3, id_{k+n}, use, uf>$와 같이 표현 가능하다.

따라서, 이러한 n_1과 n_3의 관계는 노드의 연관관계는 $sub_con\text{-}of/W_{n1,n3}$이고 이들은 모두 n_1의 상위 노드에 대한 서브노드에 포함된다. 만일 이들의 서브노드 집합 NTP가 NTP_n이라고 한다면 이들은 $NTP_n/n_1.parentNode$에 포함된다고 표현할 수 있다. 만일 n_1에 대한 자식 노드를 검색하고자 한다면 실제 검색되는 검색 노드들은 아래 식 (2.4)와 같이 표현할 수 있다.

$$ChildNodes = d_i.childNode(0) + d_i.childNode(1) + \cdots d_i.childNode(n) \quad (2.4)$$

또한, n_1에 대한 임의의 자식 노드 n_3에 대하여 n_3가 상위의 부모노드를 검색할 때에는 다음 식 (2.5)와 같은 수식을 통해 더미 노드의 상위 노드를 부모 노드로 인식한다.

$$ParentNode = d_i.parentNode \quad (2.5)$$



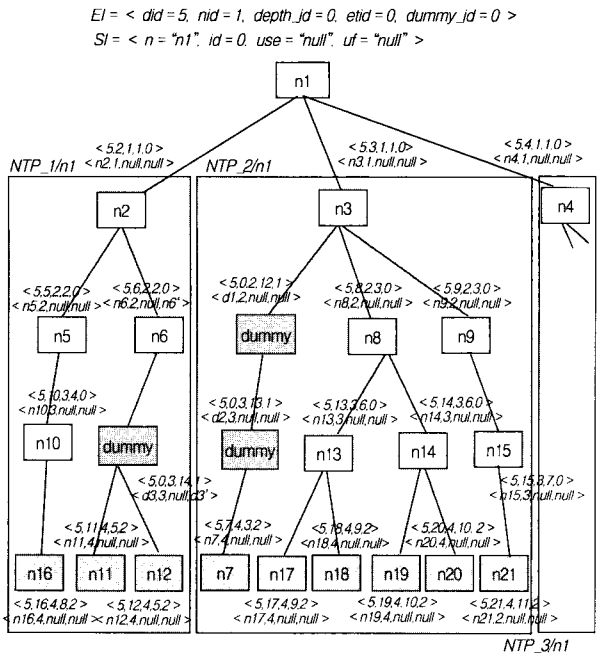
(그림 6) 더미 노드에 대한 전이적 성질표현

2.2.5 노드 분할

DODI 색인 기법에서 ***EI***와 ***SI*** 값을 할당하여 생성된 구조적 문서 트리를 관리하기 위한 최종 단계로서 본 논문에서는 최상위 노드 하위에 존재하는 노드들에 대하여 트리구조를 분할하여 최상위 노드의 서브셋(sub set) 트리구조를 별도로 저장 관리하도록 설계하였다. 분할된 서브셋 트리는 ISO 2788 표준에서 부분적 부분에 있어서의 하위 접두기호 표기방식인 **NTP(Narrow Term Partitive)**을 따르도록 설계하였다.

제안 색인 기법에서는 ***EI***와 ***SI***에 의해 할당된 구조적 문서 트리는 최상위 노드 TT(Top Term)의 NT(Narrow Term)로부터 서로 다른 NT 노드의 ETID를 갖는 하위 노드들에 대한 분할 저장, 관리하도록 설계함을 의미하는데 이에 대한 구조는 다음 그림인 구조적 문서 분할 트리구조와 같다.

구조적 문서에 대한 트리구조를 생성하고 이를 다시 분할된 하위 서브트리로 생성하는 이유는 분할된 서브트리간에 상호 연관성이 매우 적은 도메인의 특성을 지닌 경우에 매우 유용하게 정보를 제어할 수 있게 하기 위함이다. 이때 분할은 최상위 노드 TT에 대하여 서브트리는 has-a 관계를 갖게 된다. 따라서 ***n1*** 노드에 대하여 서브트리는 NTP로 구성할 수 있게된다. ***n1***에 대한 서브트리는 ***NTP_1/n1***, ***NTP_2/n1***, ***NTP_3/n1***과 같이 분할된 서브 트리 구성이 가능해진다. 이러한 분리로 인해 구조적 문서의 트리구조가 갖게 되는 방대한 양의 문서에 대한 검색과 저장을 효율적으로 관리할 수 있다. 이러한 분할을 통해 또 다른 문서의 구조적 문서 트리와의 결합을 위한 질의 재형성 과정에서도 관계정의가 용이하다.



(그림 7) DODI 구조적 문서의 분할 트리구조

3. 깊이중심분할 색인 기법을 이용한 구조적 문서 검색

3.1 시소러스에 기반을 둔 구조적 문서 검색

구조적 문서를 생성하고 여기에 더미 노드 정보와 ***SI*** 및 ***EI*** 정보 색인을 추가하여 구조적 문서 트리를 생성하여 각 노드를 객체 단위로 하여 하나의 트리에서 하나의 엘리먼트가 갖게되는 실제 정보 값인 엘리먼트 이름, 엘리먼트에 저장된 인스턴스, 엘리먼트의 각종 속성정보(***EI***에 해당)를 이용하여 시소러스의 하부저장 장치인 관계형 데이터베이스에 저장된다. 저장된 정보를 이용하여 구조적 문서에 대한 검색을 수행하도록 한다. 구조적 문서를 하부 저장 시스템에서 검색할 때에 별도로 저장된 ***SI*** 정보에 저장된 ***USE***와 ***UF*** 정보를 참조하여 실제 구조적 문서에 존재하는 노드를 찾고, 찾은 정보로부터 내용검색, 구조검색, 혼합검색 등의 시소러스 검색을 수행할 수 있다.

본 제안 색인 기법을 이용하여 검색할 항목으로는 엘리먼트의 현재 정보, 상위의 부모 엘리먼트 노드 및 하위의 자식 엘리먼트 노드를 검색하는 구조 검색을 지원하며, NTP 서브트리를 생성하도록 하여 문서의 내용을 분할 검색 가능하다. 또한 동일한 색인어에 대한 검색 및 각 계층별 엘리먼트 노드의 검색과 구조적 문서 전체에 해당하는 엘리먼트 노드의 정보를 검색 또한 가능하다.

3.1.1 검색 알고리즘

본 논문에서 제안한 시소러스를 이용한 깊이중심분할 색인 기법은 관계형 데이터베이스를 저장 시스템으로 사용한다. 기존의 관계형 데이터베이스를 사용할 경우에는 구조적 문서를 처리하기 위해 사용하는 XML 문서에 대한 대표적인 문제점을 살펴보면 다음과 같다.

첫째, 스키마 매핑의 충돌 가능성이 높다. 새로운 XML 문서를 데이터베이스로 표현하고 저장하는데는 사용자의 전처리 과정이 필요하며, 이때 XML 문서에 대한 매핑 스키마를 생성해야 한다. 그러나 이러한 매핑과정에서 기존의 관계형 데이터베이스와 XML 문서의 구문구조의 충돌이 발생할 수 있다.

둘째, 문서 변환에 따른 변환 시간이 요구된다. XML 문서 생성 시점에서 구문구조의 형식 변환이나 기존의 XML 문서에 대한 새로운 문서 형식으로의 변환과정에서 많은 수작업 시간을 요구한다.

셋째, 데이터베이스 생성 및 가공의 어려움이 있다. 데이터베이스에 존재하는 XML 문서에 대하여 XML 문서에 대한 전문에 대한 데이터베이스로의 변환되고 표현되어야 함으로 기계적으로 변환되는 정보에 대하여 사용자 질의를 유도하는데 어려움이 있기 때문이다.

본 논문에서 제안한 DODI 색인은 이러한 문제점 해결을 위해 관계형 데이터베이스에서 제공하는 저장 프로시저와 XML 문서 생성을 위해 ForXML 메소드를 사용하고, 출력을 위해 XSL를 해당 검색 항목별로 설계하여 관계형 데이터베이스와 XML 문서의 구문구조 충돌을 방지하였다. 또한 OpenXML 메소드를 이용하여 생성된 구조적 문서를 데이터베이스 저장하고, 저장된 데이터베이스로부터 분할된 구조적 문서 생성이 가능하도록 저장 프로시저를 설계하였다.

3.1.1.1 색인을 이용한 검색 알고리즘

제안한 알고리즘에서 사용하는 검색 결과는 구조적 문서 구조를 지닌 XML 문서의 정보를 DOM을 이용하여 트리구조 형태의 객체 정보를 얻을 수 있는데 이러한 트리구조의 각 객체는 하나의 엘리먼트 정보에 해당한다. 따라서 제안된 색인 알고리즘을 통한 검색은 전체 트리구조에서 원하는 엘리먼트 노드를 검색하여 해당 엘리먼트 노드의 이름이나 인스턴스를 추출해내는데 그 목적이 있다. 제안된 색인 모델을 통해서는 현재 엘리먼트 노드에 대한 부모 엘리먼트, 자식 엘리먼트, 형제 엘리먼트를 검색할 수 있다. 이는 동일한 단말 노드 깊이를 지닌 제안 트리구조에서 문서 정보인 ***dtid***와 엘리먼트 타입 정보인 ***etid***의 값, 또한 깊이 정보인 ***depth_ID***를 통해 쉽게 구할 수 있다는 장점이 있다. 다음은 제안 색인 모델링 기법을 이용하여 내용검색, 계층구조 검색 및 복합검색과 ***EI***와 ***SI***의 각 항목에 해당하는 정보에 대하여 엘리먼트의 속성 정보로 저장하여 속성 검색이 가능함을 보여준다.

① 내용 검색

내용 검색은 구조적 문서 구조가 생성한 트리구조의 엘리먼트 노드의 실제 값에 해당하는 인스턴스에 대한 검색

을 의미한다. 내용 검색을 위해서는 ***EI***의 값 중에서 엘리먼트 타입 정보인 ***etid*** 값을 추출한 다음, 키워드 인덱스로부터 질의에 해당되는 엘리먼트의 ***dtid***와 ***etid***의 집합을 추출한다. 이때 ***SI***를 통해 우선적으로 입력 질의 키워드에 대한 동의어 여부를 체크하고 해당되는 동의어가 있으면, 해당 동의어에 대한 우선어 정보를 찾아 실제 트리에 존재하는 노드를 검색한다. 만일 동일한 내용이 1개 이상일 수 있으므로 해당되는 엘리먼트 내용만큼 검색을 반복하여 수행한다.

```
(1) Let, Ci is a content element(s)
(2) c_etid := search etid from element index of EI
(3) c_set := group of EI element nodes includes Ci

(4) for (i = 1 ; i ≤ length(c_etid) ; i++) {
        for ( j = 1 ; j ≤ length(c_set) ; j++) {
            if (c_set[i].etid == c_etid ) {
                    get result_set[count++] =
                        c_set [i].nodeValue
            }
        }
    }
```

(그림 8) 내용 검색 알고리즘

② 구조 검색

구조적문서는 트리형태의 계층적 정보 구조를 갖는다. 이러한 구조적 문서에서 현재 엘리먼트 노드로부터 부모, 자식 엘리먼트 노드를 검색하는 검색 알고리즘을 의미한다. 이는 현재 엘리먼트 노드는 부모 엘리먼트 노드의 자식 엘리먼트 노드가 됨을 의미한다. 따라서 현재 기준 엘리먼트 노드의 level_up 정보를 통해 상위 노드를 검색하여 부모 엘리먼트 노드를 찾는다. level_up 속성은 구조적 문서에서는 사용되지 않으나 구조적 문서를 데이터베이스화하기 위해 DOM 트리 정보로부터 상·하위의 연관정보를 저장하기 위해 사용한다. 일반적으로 현재 엘리먼트로부터 상위 부모 엘리먼트를 여러 개 가질 수 있지만 본 논문에서 사용하는 트리구조는 상위 노드에 대하여 1개의 링크만을 갖는 구조를 기본으로 한다.

자식 엘리먼트 노드는 기준 엘리먼트 노드의 하위 노드 중에서 $etid_p$와 $depth\_ID_p$ 값을 기준으로 동일한 $etid_p$+n과 $depth\_ID_p$+1의 값을 갖는 자식 노드를 검색하면 된다. 검색 순서는 같은 ***etid***값을 갖는 자식 엘리먼트 중에 가장 작은 ***nid***값을 갖는 엘리먼트에서부터 검색한다.

형제 노드의 경우에는 ***NTP*** 내의 모든 동일 레벨을 검색하거나 하나의 문서 내에 존재하는 모든 동일 레벨을 검색할 경우에는 부모노드가 다르더라도 검색이 가능해야 하는데, 이때에는 ***EI***의 ***depth_ID***와 ***etid***를 이용하거나 ***dtid*** 추가적으로 사용한다. 먼저 검색하고자 하는 형제 엘리먼트의 ***etid***를 엘리먼트 인덱스로부터 구하고 기준 엘리먼트의 $etid_p$와 동일한 $etid_{si}$ 값을 갖는 엘리먼트 노드들을 검색하여 형제 엘리먼트를 검색한다.

③ 혼합 검색

혼합 검색은 구조적 문서 트리에서 내용과 구조를 혼합하여 검색하기 위한 알고리즘이다. 혼합 검색은 사용자의 질의 형식이 어떠한 특징을 가지고 있는가에 따라 다양하게 나타날 수 있다. 따라서 우선적으로 다른 구조 검색 방법과 마찬가지로 원하는 질의를 구하기 위해 엘리먼트의 질의를 분석하고 분석된 엘리먼트 순서정보가 우선적으로 구해져야 한다. 이후 엘리먼트 인덱스로부터 해당되는 ***nid*** 값과 ***etid***, 그리고 ***depth_ID***를 이용하여 검색 대상 엘리먼트를 파악한다. 혼합 검색에서 동의어에 관련한 엘리먼트 노드의 검색을 위해서는 ***SI*** 정보를 이용하게 되는데 이는 구조적 문서 트리에서 각 엘리먼트 노드가 갖는 정보 이외에 동의어에 관련한 정보만을 갖는 인덱스이다. ***SI***에 존재하는 ***n***(node) 값과 검색하고자 하는 키워드와 비교하여 해당되는 ***n***이 존재하면 이 ***n***의 정보가 실제 트리에 존재하는 노드인지, 아니면 ***VSN*** 인지를 구별해야 한다. 이때 사용되는 ***SI***의 속성 정보가 우선어 정보인 ***USE***값이다. 실제 트리에 존재하는 엘리먼트 노드는 동의어의 기준으로 설정하기 위해 우선어 값을 ***null***로 세팅하여 저장해 놓는다. 만일 우선어가 ***null***이 아니면 우선어에 값을 찾아 순회하면서 실제 트리에 존재하는 엘리먼트를 찾을 수 있고, 이 값이 원하는 내용 정보가 된다. 전체 구조정보를 검색하고자 할 때에는 분할된 서브트리 형식의 ***NTP***를 검색하여 전체 서브트리를 순회하여 검색할 수 있는데 이때에는 분할 ***NTP***의 최상이 엘리먼트 노드의 상위노드로부터 같은 ***etid*** 값을 갖는 노드를 모두 검색하면 가능해진다. 필요에 따라 임의의 ***n1*** 엘리먼트에 대한 ***NTP***를 ***NTP_1/n1***이라고 하면 이는 ***n1***을 ***TT***로 갖는 자손 엘리먼트 노드들의 합이라고 할 수 있다.

아래 그림은 기본 알고리즘은 다음과 같다.

```
(1) Analysing query of document elements
(2) etid_p := get current element's etid from element index
(3) EI_set := get the synonym group of N_p(EI).nodeValue
(4) SI_set := get the related group of N_p(SI) = <n, id, use, tf>
(5) Let Ci is a content element(s) // Content Search
(6) c_etid := search etid from element index of EI
(7) c_set := group of EI element nodes includes Ci
(8) for (i = 1 ; i ≤ length(c_etid) ; i++) {
        for ( j = 1 ; j ≤ length(c_set) ; j++) {
            if (c_set[i].etid == c_etid ) {
                    result_set [count++] = c_set [i].nodeValue
            }       }       }
(9) if ( dummy_IDp of N_p(EI) == 0 ) {              // BT Search
      c_etid := search etid from element index of EI
      get N_p.parentNode
```

```
(10) } else if ( dummy_IDp of Np(EI) == 1 ) {
       di = Np.parentNode
       get di.parentNode
       go to (9)
     }
```

(그림 9) 검색 알고리즘_1

```
(11) for ( (etidp == etidsi (i = 1, 2, ···, n)) of all element nodes ) {
                                                    // RT Search
       si_set := search nodes of same etid from element index of EI
                 si_count++
     }
(12) for (count until over si_count ) {
       get si[si_count++].nodeName
     }
(13) set etidchild = etidp+1, depth_IDchild = depth_IDp+1
                                                    // NT Search
     for ( (etidchild == etidi (i = 1, 2, ···, n)) of all element nodes ) {
       child_set := search nodes of same etid from element index of EI
       child_count++ }
(14) for (count until over child_count ) {
       get child [child_count++].nodeName
     }
(15) for ( ( Np (SI )[i].uf ) ≠ null ) {            // SYN Search
       if ( SIp.uf.Value == SInext.use.Value ) {
         si_set [count++] = si_set [i].n }
       get si_set [count++]. n.Value
     }
```

(그림 10) 검색 알고리즘_2

4. 깊이중심분할 색인 및 검색 기법의 구현

4.1 제안 기법의 구현 시스템

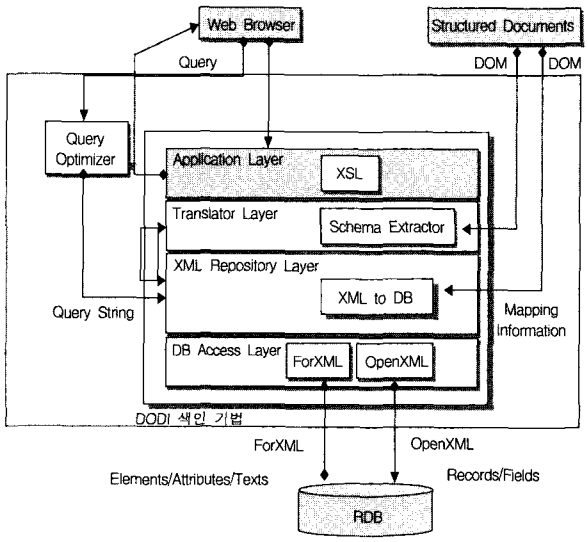
본 논문에서 제안한 DODI 색인 기법은 대량의 비 구조적인 데이터를 위한 색인 기법이 아닌 유사 계층 구조를 갖는 기능성 데이터베이스로서 트리의 깊이가 5 미만이면서 엔트리 포인트의 개수가 수 천에서 수 만개에 이르는 중소형 데이터에 대하여 그 구조 자체의 의미가 검색에서 유용한 정보로 쓰일 수 있는 구조적인 문서 형식을 저장 관리하는데 적합하도록 설계하였다. 제안 알고리즘은 하나의 구조적 문서 내에 포함된 노드에 대한 엔트리 포인트의 수가 10,000여개 내외로서 이동통신 단말기용 문자정보 서비스 및 정보 검색 서비스 등에 사용할 수 있는 정도의 응용에 적합하다. 이때 발생하는 더미노드의 수는 대략적으로 구조적 문서 트리의 skew가 발생하지 않도록 트리를 유지하기 위한 크기로서 전체 노드 수의 10%를 넘지 않는 범위에서 더미 노드가 입력되어야만 오버헤드가 크게 발생하지 않는다.

제안 색인 기법의 구현 역시 이러한 문서가 갖는 구조적 정보의 범용적인 계층성 및 계층정보의 단일화를 통한 검색 속도의 효율성 및 동일 계층의 정보에 대한 의미적 통일성을 높이는데 그 의미가 있다고 할 수 있다. 또한 동일한 깊이를 가지고 넓은 폭의 형제노드를 갖는 구조적 문헌정보의 분할을 통해 유사 항목의 문헌정보를 그룹화 하여 관리하도록 하는 정보 분할의 효율성을 높이는 데에도 그 의미가 있다.

4.1.1 시스템의 전체 구성

제안한 시소러스를 이용한 깊이중심분할 색인 기법에 대한 제안 색인 모델을 이용한 전체 시스템은 아래 (그림 11)과 같다.

제안한 색인 기법의 세부 모듈을 살펴보면 다음과 같다. 도메인 전문가에 의해 생성된 구조적 문서는 DOM을 이용하여 DB에 저장된다. 이때 DODI 색인 기법의 Translator Layer에서는 저장 프로시저를 이용하여 질의를 통해 입력된 파라미터 및 테이블 정보를 분석하여 XML Repository Layer에서는 데이터베이스에 이를 저장하게 된다.



(그림 11) 제안된 색인 모델을 위한 시스템 구성도

저장을 위해 실제 사용되는 DB 접근은 DB Access Layer에 의해 수행되는데, 이때 SQLOLEDB를 연동하여 질의에서 얻은 정보를 실제 DB에 OpenXML 메소드로 저장하게 된다. 저장된 정보는 다시 ForXML 메소드를 이용하는 DB Access Layer를 통해 필요한 템플릿을 찾아 Application Layer 사용자에게 해당 데이터베이스 정보를 XML 형식으로 변환하여 브라우징 해준다.

4.1.2 노드 색인 스키마 구조

일반적인 문서를 트리구조로 변경하고 변경된 구조적 문서 트리에서 동일한 레벨에 같은 속성의 노드를 정렬하기 위하여 더미노드를 생성하여 구조적 문서 트리에 추가하였다. 여기에 ***EI***와 ***SI*** 정보를 할당하고 여기서 얻어진 트리 구조의 검색과 저장 관리의 효율성을 높이기 위하여 TT가 NT를 기반으로 하여 엘리먼트의 속성별 분할을 수행하였

다. 이를 통해 최종적으로 시소러스를 이용한 깊이중심분할 색인 모델을 설계하였다.

이렇게 깊이 중심으로 분할된 구조적 문서 트리의 각 노드를 표현하기 위해서 ***EI***와 ***SI***의 노드 스키마를 아래와 같이 구성하여 표현하였다. ***SI***가 포함하는 정보는 크게 세 부분으로 구성되는데 노드의 타입에 관련한 정보 그룹인 ***NTI*** (Node Type Information)와 노드의 실제 값 정보를 갖는 ***NVI***(Node Value Information), 더미노드에 대한 정보를 갖는 ***DNI***(Dummy Node Information) 및 현재 노드에 대한 상위 노드 정보를 갖는 ***UNI***(Upper Level Node Information)로 구성하였다.

NTI에는 문서의 일련번호에 해당하는 문서 타입 정보인 ***dtid***와 노드 ID에 해당하는 ***nid***, 노드의 깊이 정보를 나타내는 ***depth_id***, 마지막으로 노드가 나타내려고 하는 실제 엘리먼트에 대한 타입 정보로 구성된다.

NVI는 ***NTI***의 실제 값을 저장하기 위한 스키마 구조이다. 따라서 ***NVI***의 구성은 노드에 해당하는 엘리먼트의 이름을 텍스트로 저장하는 ***n_name***, 노드에 속한 엘리먼트 내에 존재하는 실제 값을 저장하기 위한 ***n_text***, 해당 노드가 다양한 용도로 사용되기 위해 필요한 엘리먼트 속성 이름에 해당하는 ***n_attr_name***, 엘리먼트 속성에 대한 실제 값을 표현하기 위한 ***n_attr_text***로 구성하였다. 이러한 구성은 실제 구현에서 사용하게될 구조적 문서 응용을 위한 XML 문서 서식에 대한 DOM(Document Object Model)에 의해 생성되는 트리구조의 정보를 모두 포함할 수 있도록 설계하였다.

NTI	
dtid	document type
nid	node id
depth_id	level depth value
etid	element type
NVI	
n_name	TagName
n_text	node value
n_attr_name	Attr_Name
n_attr_text	Attr_value
DNI	
dummy_id	0 | 1 | 2
ULI	
level_up	parent node

(a) EI Node Schema

SYN Type Information	
n_name	TagName | dummy_id
id	Attr_id
use_id	USE(node | dummy)
use_for_id	USE FOR(node | dummy)

(b) SI Node Schema

(그림 12) EI/SI 노드의 스키마 구조_1

Document Type Definition File Information	
dtd_id	DTD id
dtd_name	dtd file name
DTD Fulltext Information	
dtd_content	dtd definition fulltext

(c) DTD Table Schema for EI

Document File Information	
dtid	document type
dtd_id	DTD id
XML File Information	
xml_content	xml fulltext

(d) Document File Schema for EI

NTP Group Information	
etid	element type id
ntp_id	NTP id
tt_node	top term of ntp_N
XML File Information	
ntp_content	xml fulltext of sub tree

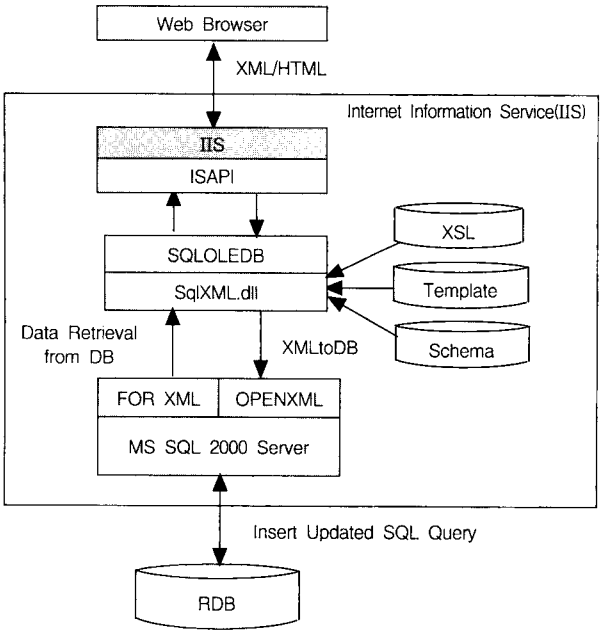
(e) NTP File Schema for EI

(그림 12) EI/SI 노드의 스키마 구조_2

DNI는 일반적인 노드 이외에 구조적 문서 트리구조에서 노드의 계층적 정보 보호를 위해 속성이 다른 노드 레벨을 정렬하기 위해 추가시킨 더미 노드에 대한 정보를 위한 일종의 플래그 값을 보관한다. 따라서 ***DNI***에는 단순한 더미 노드에 대한 플래그 정보로서 ***dummy_id***를 갖는데 이에 대한 실제 값은 0이거나 1 혹은 2인 값을 갖게된다. ***dummy_id***이 값이 0일 경우에는 구조적 문서 트리가 갖는 노드가 일반적인 노드임을 의미하며, 1의 값을 가질 경우 이는 깊이 중심의 제한적 검색을 위해 추가시킨 더미 노드임을 의미하도록 하였다. 또한 2는 모든 단말노드의 레벨 깊이를 수정한 후에 실제 인스턴스 노드 레벨인 단말노드의 검색을 빠르게 할 수 있도록 설정하기 위한 더미노드의 값이다. 따라서 ***dummy_id***이 값이 2일 경우에는 더미노드 유무에 대한 정보가 아닌 단말노드 유무를 판단하는데 사용된다.

4.1.3 XML-to-DB의 연동

본 논문에서 제안한 깊이 DODI 색인 모델링 기법에 대한 구현은 한의학 문헌정보를 사용하였다. 한의학 문헌정보는 정보 자체의 특성이 본 논문에서 제안한 구조적 문서 구조 및 색인 구조를 적용하는데 가장 적합한 형식의 컨텐츠이다. 따라서 한의학 문헌정보 중에 탕재정보에 대한 구조적 문서를 작성하고, 여기에 제안한 ***EI***와 ***SI***를 통한 색인기법을 적용하고 구조적 문서의 레벨별 엘리먼트 타입의 균형을 위해 더미노드를 삽입하여 구조적 문서를 재형성하였다.



(그림 13) ISAPI와 SQLOLEDB 연동

위의 (그림 13)의 제안 색인기법을 이용한 구조적 문서 저장 다이어그램에서 실제 데이터베이스에 XML 문서를 데이터로 저장하고 또한 데이터베이스에 저장된 데이터를 다시 브라우저를 통해 출력하기 위한 XML 문서로 변환하는데는 MS SQL Server 2000에서 제공하는 ISAPI를 이용하는데, (그림 13)은 ISAPI와 SQLOLEDB를 연동에 관한 구조이다. 그림에서 IIS 내에 존재하는 ISAPI와 SQLOLEDB의 연동을 통해 SQLXML.dll은 XML 문서 출력 및 저장형식에 관련한 XSL, XML Schema, Template을 사용하여 관계형 데이터베이스 내에 레코드 정보와 연동하게 된다. 이때 데이터베이스의 연동을 위해서는 서버에 대한 가상 루트를 설정해야 하는데, IIS 서버는 URL에 지정된 가상루트를 검사하고 이를 ISAPIDLL(sqlisapi.dll)이 가상루트에 존재하는지 유무를 확인한다.

또한 데이터베이스에 입력된 XML 문서 데이터를 ForXML 메소드를 통해 IIS는 웹브라우저에 HTML이나 XML 문서 형태로 출력하는 역할을 한다. 또한 브라우저를 통해 입력된 XML문서는 OpenXML 메소드를 통해 데이터베이스 내에 데이터로 저장하는 역할을 수행하도록 하였다. 여기에 사용되는 MS SQL Server 2000의 프로시저로는 ***sp_xml_preparedocument***와 ***sp_xml_removedocument***를 사용한다.

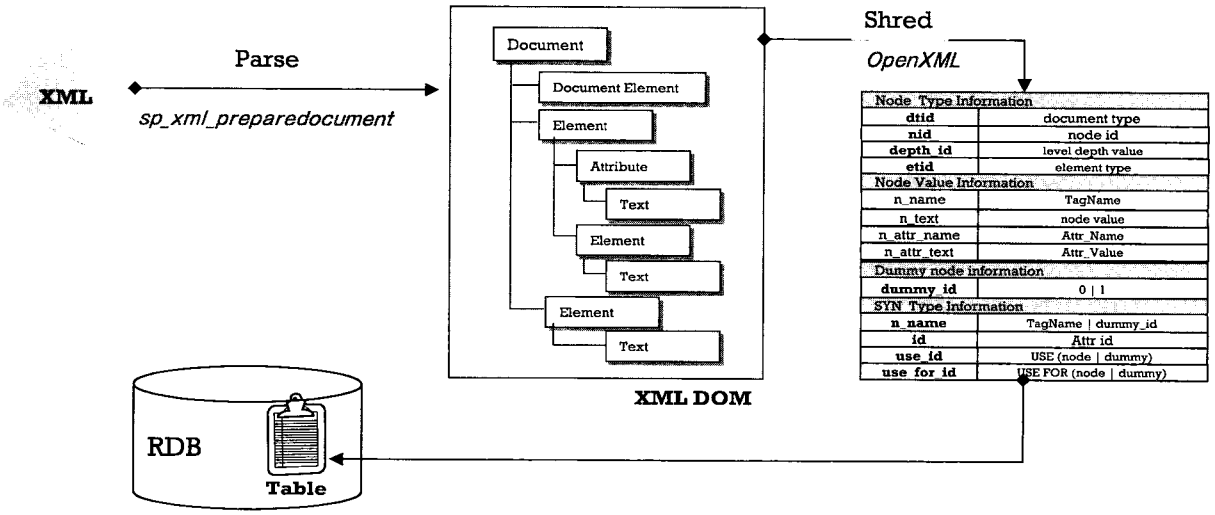
sp_xml_preparedocument는 XML 문서를 받아서 문서를 분석하고 이를 메모리에 로딩 시킨 후, 핸들 값을 반환한다. 핸들 값이 유지되는 한 메모리 상에 분석된 XML 문서는 계속 유지시키는 역할을 수행한다. 또한 OpenXML은 모든 정보의 표현을 테이블 형식으로 표현이 가능하여 실제 테이블의 칼럼(필드)정보에 XML 문서의 각 노드를 매칭시킬 수 있다. 만일 OpenXML의 사용이 되면 메모리 상에 적재 된 XML 문서를 삭제하기 위해 ***sp_xml_removedocument***를 사용한다. OpenXML의 실행은 아래 (그림 14)와 같다.

4.1.4 DODI 색인 기법의 적용

4.1.4.1 문헌정보 선정

제안 DODI 색인 기법의 적용을 위해 사용한 문헌정보는 한의학 문헌정보 컨텐츠 중 탕재정보로서 아래 (그림 15)와 같이 최대 깊이 5를 갖는다. 탕재정보에 대한 시소러스 기본 인덱스는 크게 5 단계로 구분하였다. 최상위 노드가 위치하는 TT를 0 단계로 하고, 이를 초기화단계로 설정하였다.

1단계는 구조적 문서의 트리구조에서 TT의 자식노드에 해당하는 노드로서 대분류 12개 항목의 과별 정보를 입력하였으며, 2단계는 17개의 세부 중분류를 설정하였다. 3단계는 소분류로서 2단계의 중분류 내에 존재하는 세부 분류항

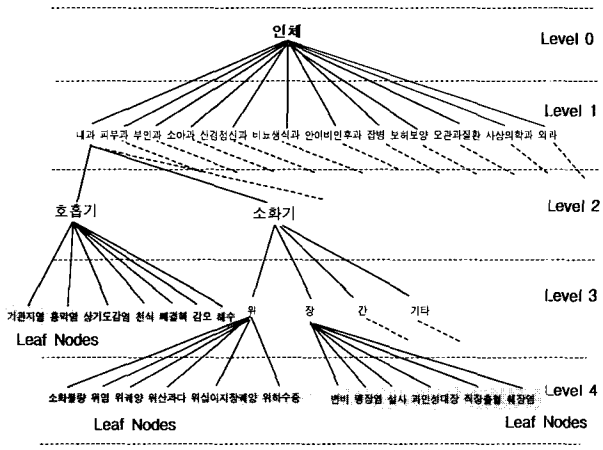


(그림 14) OpenXML의 실행 다이어그램

목에 해당한다. 마지막으로 병명별 178가지의 유형을 갖는 단말노드를 4단계로 구분하였다. 탕재에 대한 구조적 문서 트리구조는 LEVEL 0, 1, 2, 3의 경우 개념 표현 레벨에 해당하며, LEVEL 4는 실제 인스턴스에 해당하는 병명정보를 나타낸다.

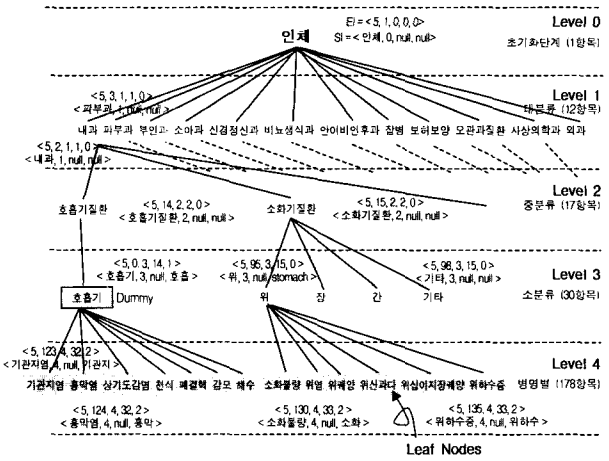
Primary	Level A	Level B	Level C	Level D	Level E	Level-Content
0	**A1**	-	-	-	-	**인 체**
1	A1	**B1**	-	-	-	**내 과**
2	A1	**B2**	-	-	-	**피부과**
3	A1	**B3**	-	-	-	**부인과**
4	A1	**B4**	-	-	-	**소아과**
5	A1	**B5**	-	-	-	**신경정신과**
6	A1	**B6**	-	-	-	**비뇨생식과**
7	A1	**B7**	-	-	-	**안이비인후과**
8	A1	**B8**	-	-	-	**잡 병**
9	A1	**B9**	-	-	-	**보허보양**
10	A1	**B10**	-	-	-	**오관과질환**
11	A1	**B11**	-	-	-	**사상의학과**
12	A1	**B12**	-	-	-	**외과**
13	A1	B1	C1	-	-	A1/B1/호흡기
14	A1	B1	C2	-	-	A1/B1/소화기
15	A1	B1	C3	-	-	A1/B1/순환기
.....						
234	A1	B11	C16	D29	E175	A1/B11/C16/D29/태음(태음인)
235	A1	B11	C16	D29	E176	A1/B11/C16/D29/태양(태양인)
236	A1	B12	C17	D30	E177	A1/B12/C17/D30/치질
237	A1	B12	C17	D30	E178	A1/B12/C17/D30/타박상

(그림 15) 문헌정보 기초색인



(그림 16) 레벨 정렬이 되지 않은 탕재의 병명별 일반 트리구조

위의 그림은 실제 문헌정보를 트리구조화 한 그림이다. 생성된 트리의 레벨의 최대 값은 5이다. 위의 문헌정보 기초색인을 바탕으로 생성한 트리는 아래 호흡기의 단말노드의 레벨이 최대 레벨과 같지 않아 level_3에 존재하는 소화기의 소분류와 중분류인 호흡기의 기관지염, 흉막염, 상기도감염, 천식, 폐결핵, 감모, 해수 등이 같은 레벨에 존재하기 때문에 계층적인 제한 검색이 불가능하다. 따라서 제안한 DODI 색인 기법을 이용하여 호흡기에 대한 더미노드를 삽입하여 색인한 구조적 문서 트리가 (그림 17)이다. DODI 색인기법을 이용한 색인을 통해 구조적 문서는 B+ 트리구조 형식을 갖게 되며, 레벨에 대한 제한적 검색을 ***EI***의 색인으로 쉽게 수행할 수 있다.



(그림 17) DODI 색인된 탕재의 병명별 구조적 문서 트리구조

4.1.4.2 DOM을 이용한 구조적 문서 트리생성

위의 그림에서와 같이 문헌정보에 대한 기초색인을 통해 본 논문에서 제안한 ***EI***와 ***SI*** 색인을 통해 다음과 같은 XML 문서를 생성할 수 있다. 아래 그림은 ***EI***와 ***SI*** 정보 색인과 레벨의 균형을 위해 더미노드를 삽입하여 색인이 완료되어 생성된 XML문서이다.

```
<?xml version = "1.0" encoding = "euc-kr"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by cache
     (cache) -->
<level_a dtid = "5" nid = "1" depth_id = "0" etid = "0" dummy_id = "0" >
     body (인체)
  <level_b dtid = "5" nid = "2" depth_id = "1" etid = "1" dummy_id = "0">
       internal department(내과)
    <level_c dtid = "5" nid = "14" depth_id = "2" etid = "2" dummy_id = "0">
         chest complaint(호흡기질환)
      <level_d dtid = "5" nid = "0" depth_id = "3" etid = "14"
           dummy_id = "1">호흡기
        <level_e dtid = "5" nid = "123" depth_id = "4" etid = "22"
             dummy_id ="0"> bronchial trouble(기관지염)
             </level_e>
        <level_e dtid = "5" nid = "124" depth_id = "4" etid = "22"
             dummy_id = "0"> pleurisy(흉막염) </level_e>
             ..... ....
        <level_e dtid = "5" nid = "135" depth_id = "4" etid = "23"
             dummy_id = "0"> wihasoo(위하수증) </level_e>
                        .....
      </level_d>
    </level_c>
  </level_b>
</level_a>
```

(그림 18) EI/SI와 더미노드를 포함하는 XML 문서

```
<!-- File Name : ShowNodes.htm -->
<HTML>
<HEAD> <TITLE> Show DOM Nodes </TITLE>
 <SCRIPT LANGUAGE = "JavaScript" FOR = "window"
  EVENT = "ONLOAD">
   /* get Document node : */
   Document = dsoXML.XMLDocument;
   /* start by passing the Document node to DisplayNodes : */
   DisplayDIV.innerText = DisplayNodes(Document, 0);
   function DisplayNodes (Node, IndentLevel)
      {
      /* declare local variables for recursion : */
      var i;
      var DisplayString = " " ;
      /* build up the indentation for this level : */
      Indent = " " ;
      IndentDelta = "      " ;
      for (i = 0 ; i < IndentLevel ; ++i)
          Indent += IndentDelta;
      /* display the current node's properties : */
      DisplayString += Indent + "nodeName : " + Node.nodeName + "\n"
                    + Indent + "nodeValue : " + Node.nodeValue
                    + "\n\n" ;
      /* display the current node's properties : */
      DisplayString += Indent + "nodeName : " + Node.nodeName + "\n"
                    + Indent + "nodeValue : " + Node.nodeValue
                    + "\n\n" ;
      /* display each of the node's attribute child nodes : */
      Indent += IndentDelta;
      for (i = 0 ; Node.attributes != null && i < Node.attributes.length ; ++i )
      DisplayString += Indent + "nodeName : "
                    + Node.attributes(i).nodeName + "\n" + Indent
                    + "nodeValue : " + Node.attributes(i).nodeValue
                    + "\n\n" ;
      /* display each of the node's nonattribute child nodes : */
      for (i = 0 ; i < Node.childNodes.length ; ++i )
      DisplayString += DisplayNodes (Node.childNodes(i), IndentLevel + 1) ;
      /* return the string containing the results : */
      return DisplayString ;
      }
  </SCRIPT> </HEAD>
<BODY>    <XML ID = "dsoXML" SRC = "hanbang_3.xml"> </XML>
          <H2> XML Document Object Model (DOM) Nodes
          </H2> <DIV ID = "DisplayDIV"> </DIV>
</BODY> </HTML>
```

(그림 19) DOM을 이용한 구조적 문서의 출력

4.1.4.3 저장 프로시저를 이용한 XML 문서의 저장

생성된 XML 문서를 관계형 데이터베이스 내에 저장하기 위해서는 OpenXML 메소드를 이용하였다. 이를 통해 구조적 문서의 엘리먼트 요소, 텍스트, 속성 등의 정보를 데이터베이스 테이블에 저장한다. 구조적 문서에 대한 OpenXML 메소드를 다음과 같이 저장 프로시저로 생성하여 사용한다.

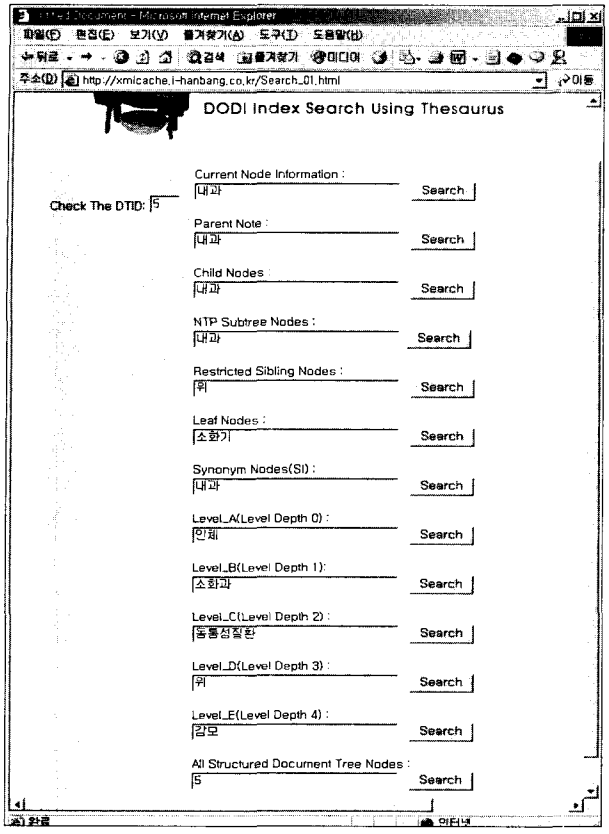
```
CREATE PROC tempxmldoc_a @xmldoc ntext
AS
DECLARE @hwd int
EXEC sp_xml_preparedocument @hwd output, @xmldoc

INSERT INTO EI_level_a
SELECT * from openxml(@hwd, 'level_a', 3)
with (NodeName varchar(50) '@mp : localname', NodeText varchar(50)
      'text( )', etid
varchar(50), level_up varchar(50), dtid varchar(50), nid varchar(50), depth_id
varchar(50), dummy_id int)
EXEC sp_xml_removedocument @hwd

// Using Stored Procedure
EXEC tempxmldoc_a '<level_a dtid = "5" nid = "1" depth_id = "0" etid = "0"
                   dummy_id = "0"> body(인체)
  <level_b dtid = "5" nid = "2" depth_id = "1" etid = "1" dummy_id = " 0">
        internal department(내과)
   <level_c dtid = "5" nid = "14" depth_id = "2" etid = "2" dummy_id = "0">
         chest complaint(호흡기질환)
    <level_d dtid = "5" nid = "0" depth_id = "3" etid = "14" dummy_id = "1">
          호흡기
     <level_e dtid = "5" nid = "123" depth_id = "4" etid = "22" dummy_id = "2">
          bronchial trouble(기관지염)
     </level_e>
     <level_e dtid = "5" nid = "124" depth_id = "4" etid = "22" dummy_id = "2">
          pleurisy(흉막염) </level_e>
     <level_e dtid = "5" nid = "125" depth_id = "4" etid = "22" dummy_id = "2">
          respiratory tract (상기도감염) </level_e>
     <level_e dtid = "5" nid = "126" depth_id = "4" etid = "22" dummy_id = "2">
          asthma(천식) </level_e>
                  ... 중략 .....
  <level_b dtid = "5" nid = "13" depth_id = "1" etid = "1" dummy_id = "0">
        surgery(외과)
   <level_c dtid = "5" nid = "0" depth_id = "2" etid = "13" dummy_id = "1">
         surgery complaint(외과질환)
    <level_d dtid = "5" nid = "0" depth_id = "3" etid = "13" dummy_id = "1">
          외과
     <level_e dtid = "5" nid = "93" depth_id = "4" etid = "13" dummy_id = "2">
          치질 </level_e>
     <level_e dtid = "5" nid = "94" depth_id = "4" etid = "13" dummy_id = "2">
          타박상 </level_e>
    </level_d>
   </level_c>
  </level_b>
</level_a>
```

(그림 20) 저장프로시저 내의 OpenXML 질의

4.1.4.4 검색 항목 선정 및 검색 결과

저장된 구조적 문서의 노드 검색을 위해 13가지의 구조적 문서 검색 결과항목을 선정하였다. 이는 현재 검색을 원하는 입력 키워드에 대한 ***SI***를 검색하여 동일한 ***SI*** 색인이 존재하면 해당 ***SI***의 우선어를 찾아 동일한 ***dtid***를 갖는 실제 ***EI*** 색인에 존재하는 엘리먼트 노드를 검색한다.

13가지 검색 항목에는 현재 노드, 입력 키워드에 대한 부모노드, 자식노드, 시소러스에 의한 XML 분할 트리인 NTP 서브트리, 동일한 부모노드를 갖는 제한적 형제노드, 실제 인스턴스 노드인 단말노드, 동의어 노드, 각 계층별 전체 노드 및 동일 ***dtid***에 존재하는 모든 노드인 전체 XML 문서를 검색한다. 다음 (그림 21)은 이에 해당하는 검색 항목 리스트를 보여주고 있다.



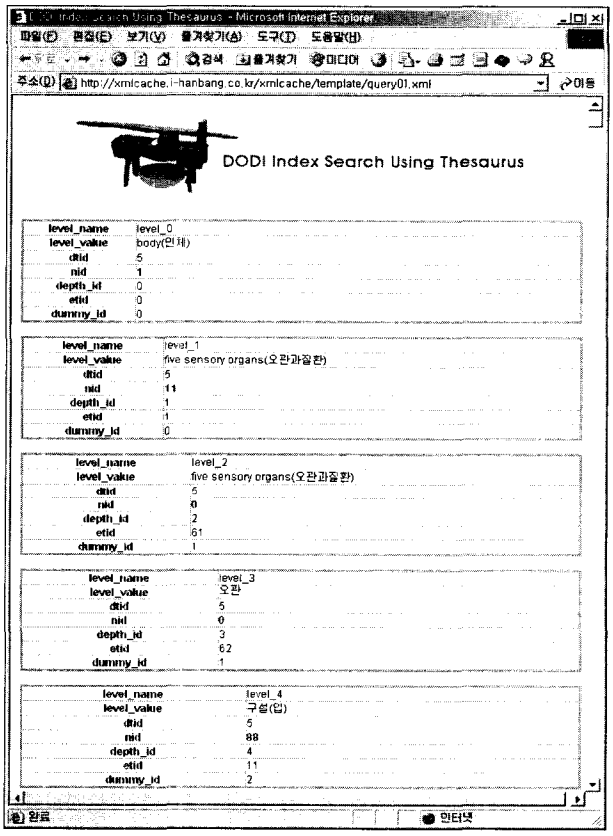
(그림 21) 검색항목의 선정

(그림 22) template를 이용한 저장프로시저의 XML 문서 생성 예

이때 ForXML 메소드를 이용하는데 ForXML 구문은 크게 4가지의 인수를 갖는다. XML 기본 구문의 mode는 RAW, AUTO, EXPLICIT 중 하나의 모드를 사용하는데 RAW와 AUTO 모드는 XML 구문의 세부적인 엘리먼트에 대한 커스터마이징이 어렵기 때문에 EXPLICIT 모드를 이용하여 보다 세부적인 XML 문서의 트리를 제어할 수 있다.

검색 항목에 입력된 키워드가 *SI* 색인으로부터 실제 노드를 검색하여 생성한 전체 XML 문서의 출력은 (그림 22)과 같다.

ForXML 메소드를 템플릿을 통해 XML 문서로 생성한 후에는 이를 사용자가 원하는 출력 형식에 맞는 출력서식을 설정해야 한다. 이를 위해 스타일 시트 언어인 XSL을 사용하였다. 출력을 위한 XSL을 통해 얻어진 출력 정보는 (그림 23)과 같다.



(그림 23) XSLT를 이용한 DODI 색인 XML 문서 생성 예

4.1.4.5 XML 문서의 생성

DOM은 문헌정보 데이터를 데이터베이스에 저장하기 위해 OpenXML 메소드를 사용하기 이전에 OpenXML 메소드를 통해 입력되는 텍스트형식의 정보를 DOM을 통해 얻을 수 있으며, 검색 프로시저가 완성됨과 함께 ForXML 및 OpenXML, 그리고 DOM의 처리 과정에 대한 응용결과를 삽입하도록 하였다. (그림 24)는 ForXML의 Explicit를 이용하여 데이터베이스 내에 저장된 구조적 문서정보 전체를

XML 문서 트리로 생성하기 위하여 저장프로시저에 저장한 질의어이다.

여기서 기본적인 Explicit 문의 파라미터는 4가지로 다음 식(4.1)과 같이 구분할 수 있다

ELEMENTName!TagNumber!AttributeName!Directive (4.1)

여기서 ***ELEMENTName***은 엘리먼트의 이름을 추출하는데 사용되며 ***TagNumber***는 엘리먼트 태그의 번호, ***AttributeName***은 엘리먼트의 속성을 지정하는데 사용한다. 또한 Directive는 부가선언 시 사용되는 파라미터이다.

데이터베이스 내에 저장된 테이블로부터 XML 문서 생성을 위한 저장 프로시저에서 "***1 as Tag, Null as Parent***"의 선택 필드 출력의 의미는 저장된 테이블에서 가상 상위의 루트노드에 대하여 그에 이전 상위 노드가 존재하지 않음을 표현하기 위해 루트 노드의 상위 노드 필드 출력에 대하여 Null을 출력하도록 유도한다. 또한 각 서브 엘리먼트에 대한 구문을 선택하고 결합하여 최종적으로 ***'ORDER BY [level_b!1], [level_c!2], [level_d!3], [level_e!4]'***를 통해 엘리먼트를 정렬하여 출력하도록 하였다.

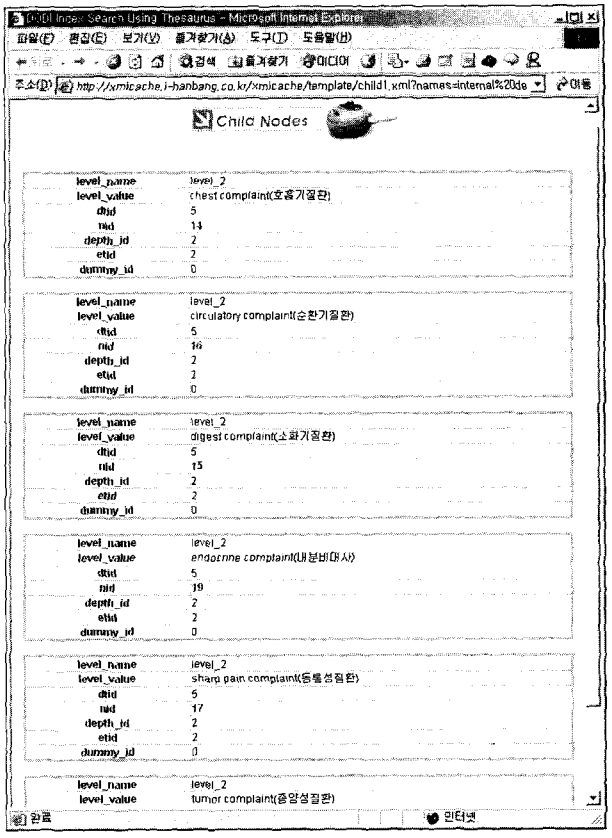
```
SELECT DISTINCT 1 as Tag, Null as Parent, EI_levela.level_value as
[level_a!1], EI_levela.dtid as
[level_a!1!dtid], EI_levela.nid as [level_a!1!nid], EI_levela.depth_id as
[level_a!1!depth_id], EI_levela.etid as [level_a!1!etid], EI_levela.dummy_id as
[level_a!1!dummy_id],
Null as [level_b!2], Null as [level_b!2!dtid], Null as [level_b!2!nid], Null as
[level_b!2!depth_id], Null as [level_b!2!etid], Null as [level_b!2!dummy_id],
Null as [level_c!3], Null as [level_c!3!dtid], Null as [level_c!3!nid], Null as
[level_c!3!depth_id], Null as [level_c!3!etid], Null as [level_c!3!dummy_id],
Null as [level_d!4], Null as [level_d!4!dtid], Null as [level_d!4!nid], Null as
[level_d!4!depth_id], Null as [level_d!4!etid], Null as [level_d!4!dummy_id],
Null as [level_e!5], Null as [level_e!5!dtid], Null as [level_e!5!nid], Null as
[level_e!5!depth_id], Null as [level_e!5!etid], Null as [level_e!5!dummy_id]
FROM EI_levela

union all

SELECT DISTINCT 2,1, EI_levela.level_value, Null, Null, Null, Null, Null,
EI_levelb.level_value, EI_levelb.dtid, EI_levelb.nid, EI_levelb.depth_id,
    EI_levelb.etid,
EI_levelb.dummy_id,
Null, Null, Null, Null, Null, Null,
Null, Null, Null, Null, Null, Null,
Null, Null, Null, Null, Null, Null
from EI_levela, EI_levelb
where EI_levela.level_value = EI_levelb.level_up
                    ..... 중략 .....

union all
SELECT DISTINCT 4,3, EI_levelb.level_value, Null, Null, Null, Null, Null,
EI_levelc.level_value, Null, Null, Null, Null, Null,
EI_leveld.level_value, Null, Null, Null, Null, Null,
EI_levele.level_value, EI_levele.dtid, EI_levele.nid, EI_levele.depth_id,
    EI_levele.etid, EI_levele.dummy_id
from EI_levelb, EI_levelc, EI_leveld, EI_levele
where EI_levelb.level_value = 'skin(피부과)' and EI_levelb.level_value
    = EI_levelc.level_up and
EI_levelc.level_value = EI_leveld.level_up and EI_leveld.level_value
    = EI_levele.level_up
ORDER BY [level_b!1], [level_c!2], [level_d!3], [level_e!4]
FOR XML EXPLICIT
```

(그림 24) 저장프로시저 내의 ForXM

4.1.4.6 DODI를 이용한 검색 예

① 입력 노드(내과) 에 대한 Child SI 검색 및 EI 노드 정보 검색

(그림 21)에서 입력된 키워드에 대한 ***SI*** 색인 검색을 통해 입력된 키워드(내과)의 ***EI***를 검색하고, 여기에 해당하는 하위의 자식 노드들을 검색하여 chest complaint(호흡기질환), *circulatory complaint*(순환기질환), *digest complaint*(소화기질환), endocrine complaint(내분비대사), sharp pain complaint(동통성질환), tumor complaint(종양성질환) 등 6개의 자식노드를 검색한 결과가 (그림 25)와 같다.

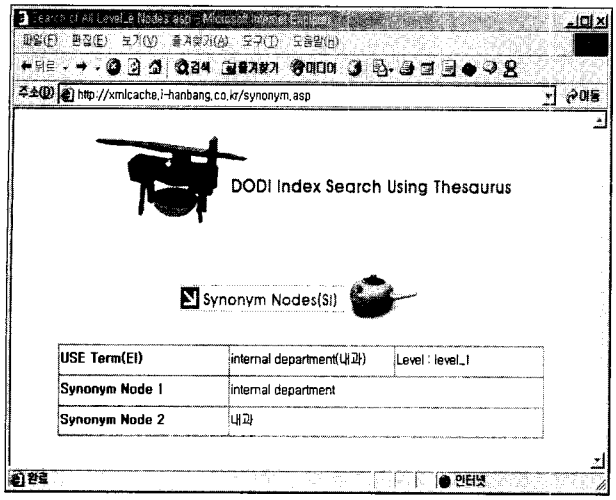


(그림 25) 입력 노드(내과)에 대한 자식 노드 검색 결과

② 동의어 노드 검색

(그림 21)에서 입력된 키워드에 대한 Synonym Nodes (SI) 검색은 입력된 키워드에 대하여 ***SI*** 색인 검색을 통해 입력된 키워드(내과)의 우선어(USE)와 비우선어(UF)의 값을 찾아 연결된 ***SI*** 색인 노드를 검색하는데, 이를 통해 얻

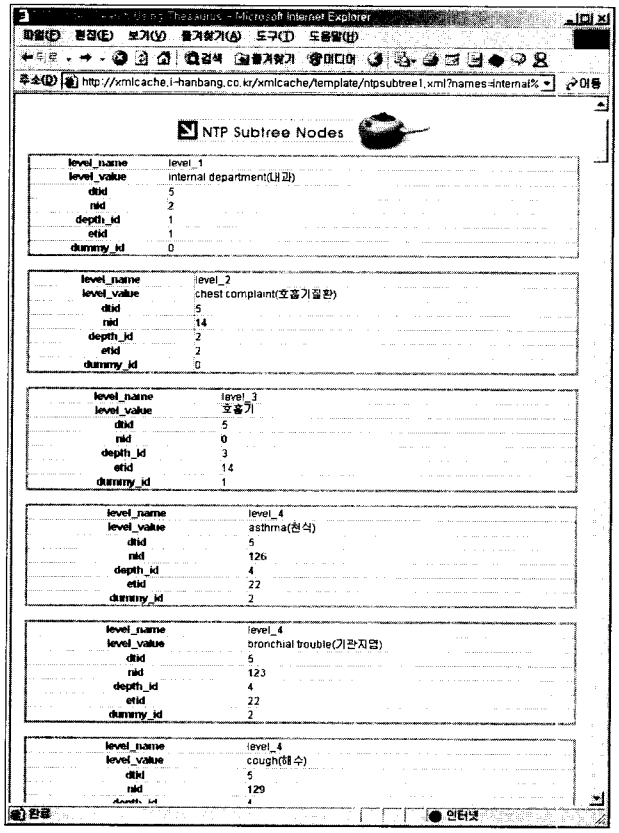
은 동의어 검색 결과는 (그림 26)과 같다



(그림 26) Synonym Node의 검색 결과 출력

③ 입력 노드(내과) 에 대한 NTP Subtree Node 정보 검색

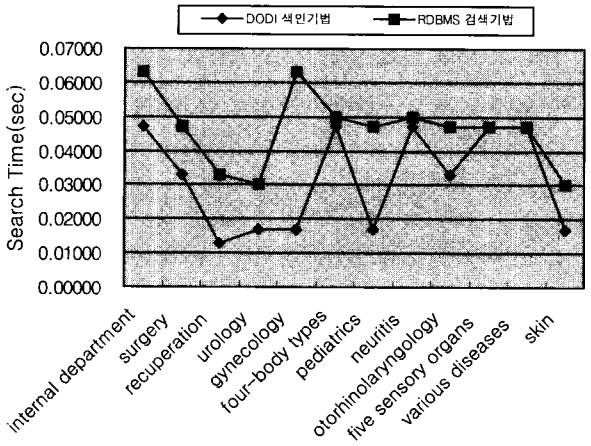
(그림 21)에서 입력된 키워드에 대한 NTP 서브 트리 노드 검색은 ***SI*** 색인 검색을 통해 입력된 키워드(내과)의 ***EI***를 검색하고, 해당 ***EI***를 조상으로 하는 모든 하위의 자식 노드를 단말노드까지 검색한다. 이를 통해 얻은 NTP 서브 트리의 검색 결과는 (그림 27)과 같다.



(그림 27) NTP Subtree의 검색 결과 출력

4.3.3 성능평가

성능 평가부분은 현재 OpenXML 메소드를 통해 저장된 구조적 문서를 ForXML 메소드를 통해 NTP 서브트리 형식의 XML 문서를 생성하는데 걸리는 시간과 일반 RDBMS를 이용하여 데이터베이스 내에 존재하는 NTP 서브트리에 해당되는 노드를 모두 검색할 때 걸리는 시간을 비교하였다. DODI 색인 알고리즘을 이용하면 ForXML을 사용하여 NTP 서브트리를 XML 형식으로 완벽하게 구현가능하며 일반적인 RDBMS의 질의 방식을 이용한 조인 방법보다 검색 속도가 빠르게 나타남을 알 수 있다.



(그림 28) DODI 색인기법을 이용한 검색 시간 비교

성능평가에 사용된 문헌정보는 한의학 탕재에 관련한 구조적 문서 생성에 사용된 엘리먼트 노드의 수는 237개이며 2,370개의 EI 정보 색인과 5,688개의 속성의 이름 및 속성 값을 가진 SI 정보 색인을 이용하였다. (그림 28)은 검색시간을 비교한 결과인데, DODI 색인 기법의 경우 평균 검색 시간이 0.3183이며 RDBMS를 이용한 평균 검색 시간은 0.4617로서 DODI 색인 기법이 약 31% 정도의 검색시간이 빠르게 이루어져 속도향상을 이루었다. 이러한 NTP 서브트리 생성은 해당 자손 노드가 많을수록 검색 시간 DODI 색인 기법이 현저히 빠른 검색 결과를 출력하게 됨을 보여주었다.

이는 NTP 서브트리를 현재는 level depth 1을 기준으로 생성하였을 때를 비교하였으며, 만일 더 낮은 계층에서 하위의 NTP를 생성할 경우에도 제안 색인 기법의 검색 속도 향상을 가져올 수 있다.

또한 검색속도 비교에 있어서도 DODI 색인에서는 NTP 서브트리의 모든 엘리먼트 노들에 대한 검색뿐만 아니라 이를 XML 문서로 생성하는 과정까지의 시간을 나타내며, RDBMS의 일반 검색에서는 엘리먼트 노드에 대한 검색 시간만을 출력해 내었으며, 이러한 노드 검색에서는 XML 구조적 문서 생성이 이루어지지 않은 상태이므로 DODI 색인

의 성능이 보다 향상된 검색 속도를 보여준다.

5. 결 론

본 논문에서는 시소러스를 이용한 구조적 문서의 탐색을 위한 깊이중심분할 색인 기법을 통해 검색 시스템에서 효과적인 검색을 위해 사용되는 색인과 검색 기능에 대한 기법을 제안하였다. 시소러스는 검색 시스템의 재현율과 정확율을 향상시키기 위해 필수적으로 요구되는 도메인 지식으로 사용되는데, 시소러스를 이용한 관리시스템을 이용하면 이러한 도메인 지식을 효과적으로 구축하고 일관성 있게 유지할 수 있도록 한다는 점에서 매우 중요하다.

이를 위해 객체를 기반으로 하는 구조적 문서에 대한 저장과 검색을 위해 XML 문서 형식을 이용하여 시소러스 관리시스템을 설계하고 구현하였다. 특히, 구조적 문서 생성을 위해 생성한 트리구조에 대하여 트리에 의해 생성된 노드 엘리먼트의 계층적 정보를 효율적으로 관리하기 위해 계층적 트리를 노드의 속성에 따라 서브트리로 분할하고 각 노드는 레벨에 따라 동일한 타입의 속성을 갖도록 더미노드를 추가하여 구조적 문서 트리구조를 재형성하였다.

제안한 깊이중심분할 색인 기법을 위해 앞서 언급한바와 같이 일반적인 문서를 트리구조로 변경하고 변경된 구조적 문서 트리에서 동일한 레벨에 같은 속성의 노드를 정렬하기 위하여 더미노드를 생성하여 구조적 문서 트리에 추가하였다. 여기에 ***EI***와 ***SI*** 정보를 할당하고 여기서 얻어진 트리구조의 검색과 저장 관리의 효율성을 높이기 위하여 TT가 NT를 기반으로 하여 엘리먼트의 속성별 분할을 수행하였다. 이를 통해 최종적으로 얻어진 깊이중심분할 색인 모델을 설계하였다. 이렇게 깊이 중심으로 분할된 구조적 문서 트리의 각 노드를 표현하기 위해서 ***EI***와 ***SI***의 노드 스키마를 아래와 같이 구성하여 표현하였다. ***SI***가 포함하는 정보는 크게 세 부분으로 구성되는데 노드의 타입에 관련한 정보 그룹인 ***NTI***(Node Type Information)와 노드의 실제 값 정보를 갖는 **NVI**(Node Value Information), 더미노드에 대한 정보를 갖는 ***DNI***(Dummy Node Information)으로 구성하였다. 또한 더미노드에 대하여 더미노드 상위노드와 더미노드의 하위노드들에 대한 속성을 연결하기 위한 전이적 성질을 이용한 시소러스 구축 메커니즘을 통해 시소러스 관리 시스템의 구조를 파악할 수 있게 한다. 이를 통해 파악된 도메인 정보에 대하여 전문가에게 필요에 따라 제공할 수 있는 시소러스 구축정보에 의해 시소러스의 일관성을 유지시킬 수 있다. 또한 구축 도메인 구축전문가가 각각의 개념에 따른 개별적 관계를 명시할 필요가 없어 구축비용을 최소화 할 수 있다.

시소러스를 이용한 구조적 문서의 탐색을 위한 깊이중심분할 색인 기법은 질의 방식에 있어서 해당 엘리먼트 노드에 대하여 일반적인 개념에서부터 구체적인 의미로의 시소러스 개념 질의 방식을 사용하기 때문에 상위 노드나 하위 노드에 대한 참조가 가능하고 각 레벨에 존재하는 노드는 더미노드 삽입을 통해 유사 속성으로 구성됨으로서 각 레벨에 대한 깊이 정보만을 이용하여 간단한 검색이 가능하다. 또한 더미노드는 검색을 위한 트리구조의 균형을 유지하고 개별적인 연관관계의 명시 없이도 트리구조를 사용할 수 있는 XML 문서에 적합하다. 또한 이러한 검색 기법은 일반적인 관계형 데이터베이스검색에 비하여 검색 속도가 향상됨을 보였다.

향후 연구과제로는 특정 도메인 정보에 대한 제안된 시소러스 색인 기법을 통해 검색 시스템에 적용함에 있어서 검색 성능을 보다 정량적으로 평가해야 할 수 있는 방법을 제시하여야 한다. 또한 이러한 평가를 통해 제안된 시소러스에 묵시적으로 설정되어지는 초기 가상 동의어 값에 대한 수를 일관성 있게 조절할 수 있도록 해야한다. 또한 구조적 문서의 ***NTP*** 서브트리를 이용한 이동통신 등의 소규모 문헌정보 서비스에 동일계층 정보를 일반화하는 응용서비스 등에 적용이 필요하다.

참 고 문 헌

[1] M. Gorden and P. Pathak, "Finding information on the World Wide Web : the Retrieval Effectiveness of Search Engines," Information Processing and Management, Vol. 35, No.2, pp.141-180, 1999.

[2] H. Chen and V. Dhar, "Online Query Refinement on Information Retrieval Systems : A Process Model of Searcher/System Interactions," In Proceedings of the 13th Annual International ACM/SIGIR Conference, Brussels, Belgium, pp.115-133, 1990.

[3] G. Salton and C. Buckley, "Improving Retrieval Performance by Relevance Feedback," Journal of the American Society for Information Science, Vol.41, No.4, pp.288-297, 1990.

[4] R. Davis, *Intelligent Information System* : Progress and Prospects, Ellis Horwood, 1986.

[5] 류성호, "구조검색을 위한 XML 문서 저장 시스템의 설계 및 구현", 배제대학교 석사학위논문, 2000.

[6] M. Hancock-Beaulieu M. Fieldhouse and T. Do, "An Evaluation of Interactive Query Expansion in an Online Library Catalogue with a Graphical User Interface," *Journal of Documentation*, Vol.5, No.3, pp.225-245, 1995.

[7] J. Y. Nie and M. Brisebois, "An Inferential Approach to Information Retrieval and its Implementation using a Manual Thesaurus," Artificial Intelligence Review, Vol.10, No.5, pp.409-439, 1996.

[8] H. J. Peat and P. Willett, "The Limitation of Term Co-occurrence Data for Query Expansion in Document Retrieval System," Journal of the American Society for Information Science, Vol.42, No.5, pp.378-383, 1991.

[9] D. A. Krooks and F. W. Lancaster, "The Evolution of Guidelines for Thesaurus Construction," Libri, Vol.43, No. 4., pp.326-342, 1993.

[10] Y. Qiu, "Automatic Query Expansion Based on a Similarity Thesaurus," Ph. D. Thesis, ETH Zurich, Institute of Computer Systems, 1995.

[11] U. Miller, "Thesaurus Construction : Problems and their Roots," Information Processing and Management, Vol.33, No.4, pp.481-494, 1997.

[12] Dario Lucarella, Antonella Zanti, "A Visual Retrieval Environment for Hypermedia Information Systems," ACM Transactions on Information Systems, Co.14, No.1, pp. 3-29, 1996.

[13] *POET : Content Management Suite 2.0*, POET Software, http://www.poet.com, 1999.

[14] K. Shoens, A. Luniewshki, P. Schwarz, J. Stamos, J. Thomas, "The Rufus System : Information Organization for Semi-Structured Data," In Proceedings of the International Conference on Very Large Database(VLDB), pp.97-107, 1994

[15] *eXcelon*, eXcelon Corporation, http://www.odi.com, 1999.

[16] *Tamino*, Software AG, http://www.softwareag.com, 1999.

[17] K. Wang, L. Huiging, "Discovering Typical Structures of Documents : A Road Map Approach," ACM SIGIR conference R&D in IR, 1998.

[18] 빈진영, "구조 기반 검색을 지원하는 XML DTD 데이터베이스의 설계 및 구현", 단국대학교 대학원 석사학위논문, 1999.

[19] 박종관, "XML 문서의 효율적인 구조 검색을 위한 색인 모델", 충북대학교 대학원 석사학위논문, 2001.

[20] 김성욱, "DOM 인터페이스를 이용한 XML 문서 저장시스템의 설계 및 구현", 한국외국어대학교 대학원, 2001.

양 옥 렬

e-mail : cache@wonkwang.ac.kr

1995년 원광대학교 컴퓨터공학과(학사)

1997년 원광대학교 대학원 컴퓨터공학과 (석사)

1999년 원광대학교 대학원 컴퓨터공학과 (박사)

1998년~2000년 BNS Media Tech. 대표

2000년~현재 군장대학 겸임교수

2000년~현재 (주)휴먼미디어테크 연구소장

관심분야 : XML, HCI, 멀티미디어 데이터베이스, 멀티미디어 저자도구, 원격교육, 멀티모달 응용

이 용 주

e-mail : yjlee@wonkwang.ac.kr

1976년 고려대학교 전자공학과(학사)

1980년~1994년 한국전자통신연구소 자동통역연구실 실장(책임연구원)

1987년 고려대학교 대학원 전자공학과 (석사)

1992년 고려대학교 대학원 전자공학과(박사)

1994년~현재 원광대학교 공과대학 컴퓨터 및 정보통신 공학부 교수

관심분야 : HCI, 음성합성, 음성인식, 음성DB, 복지공학, 멀티미디어시스템