확장된 소프트웨어 컴포넌트 서술자에 기초한
컴포넌트 저장소의 검색

금영욱*** 박병섭**

요 약

컴포넌트 저장소의 효율적인 검색이 컴포넌트 재사용에 매우 중요하다. 컴포넌트 저장소에 보관된 컴포넌트에 대한 정보를 얻는데 일반적으로 많은 시간과 노력이 필요하다. CORBA 3의 소프트웨어 컴포넌트 서술자라는 XML을 사용하여 일반적인 컴포넌트의 특성을 기술한다. 본 논문에서는 CORBA 3의 소프트웨어 컴포넌트 서술자를 확장하여 이를 사용하여 컴포넌트 저장소의 검색에 필요한 정보를 얻는다. 죽식에 기초한 새로운 검색 방법을 제안하여 기존의 죽식 방법에서 지원하지 않았던 논리 연산자를 사용한 검색이 가능하며 또한 검색의 복잡도가 확장된다.

Component Retrieval using Extended Software Component Descriptor

Young Wook Keum*** Byoung Seob Park**

ABSTRACT

Components are stored in a component repository for later reuse. Effective search and retrieval of desired components in a component repository is a very important issue. It usually takes a lot of time and efforts to gather information about a component, and its availability is essential to implement a repository. Software Component Descriptor proposed in CORBA 3 contains information about a component using an XML vocabulary. In this paper we extend Software Component Descriptor to be useful for the search of a component repository. We use a facet scheme as a search method of a component repository. And our new retrieval method supports queries connected with logical operators such as AND, OR, NOT, which were not supported with existing facet retrieval methods. Also we reduce the search complexity considerably.

키워드: 컴포넌트 저장소(component repository), 패시트(facet), 소프트웨어 컴포넌트/software component), 소프트웨어 컴포넌트 서술자/software component descriptor), 검색 방법(retrieval method)

1. 서론

소프트웨어 개발의 복잡성과 가격의 대폭적으로 증가하면서 80년대 중반 이후 효율적인 소프트웨어 재사용이 점점 중요하게 되었다. 소프트웨어 재사용을 위한 현제까지 알려진 최선의 방법은 컴포넌트에 기반한 소프트웨어를 만드는 것이다. 컴포넌트에 기반한 소프트웨어 개발을 위해서 해결해야 하는 중요한 문제 중 하나는 검색 문제이다. 컴포넌트 저장소에서 원하는 소프트웨어 컴포넌트를 효율적으로 검색하는 방법이 필요하다[3].

컴포넌트 저장소에서 컴포넌트를 검색하는 방법에 대한 많은 연구가 있었으며 상용화된 컴포넌트 저장소의 제공도 다양하다[4].

컴포넌트 저장소에 있는 컴포넌트를 효율적으로 검색하려면 먼저 컴포넌트에 관한 오타와 정보를 가지고 있어야 한다. 그러나 컴포넌트에 관한 정보를 지식한 인터페이스에 관한 정보의 수만큼 많은 어려움이 있으며 컴포넌트에 관한 정보를 얻는 방법이 컴포넌트의 종류에 따라 다르며 복잡한 프로그래밍을 통해 가능하다[5]. CORBA 3의 소프트웨어 컴포넌트 서술자(Software Component Descriptor)와 같이 컴포넌트를 표현하는 방법이 일부 존재하지만 컴포넌트 검색은 활용하고 있지 않는다.

이에 본 논문에서는 컴포넌트에 대한 정보를 얻는 효율적인 방법을 제안하고 이를 사용하여 컴포넌트를 표현하고 또한 효과적인 컴포넌트 검색에 사용한다. CORBA 3의 소프트웨어 컴포넌트 서술자는 비교적 컴포넌트에 대한 상세한 정보를 가지고 있다. 그러나 가장 중요한 정보인 인터페이스에 대한 표현이 미약하며 인터페이스 표현을 할 수 없는 컴포넌트의 기능에 대한 추가의 정보가 필요하다.

이 논문의 기여하는 바는 크게 세 가지이다. 첫 번째로 컴포넌트의 인터페이스에 대한 정보를 XML 인터페이스 표현.
하여 CORBA 3에서 제안된 소프트웨어 컴포넌트 서술자를 확장한다. 컴포넌트의 제작자나 컴포넌트와 확장된 소프트웨어 컴포넌트 서술자를 함께 제공할 수 있도록 유동식으로 컴포넌트 사용자가 컴포넌트에 대한 정보를 쉽게 얻을 수 있다. 확장된 서술자는 CORBA 컴포넌트 외에 다른 컴포넌트들과 일반 프로그래밍에도 사용 가능하다.

두 번째로 확장된 소프트웨어 컴포넌트 서술자에서 자료들을 추출하여 컴포넌트 검색에 이용한다. 컴포넌트 검색을 위한 방법으로 패턴(facet)[9-11]을 사용하는데 확장된 소프트웨어 컴포넌트 서술자에서 추출한 자료들은 패턴으로 정합이 용이하다.

세 번째로 패턴 검색 방법으로 신경 촉각 헤프[11]을 확장하여, 다른 패턴 검색[9-12]에서는 기본 검색만 제공하였지만 본 논문에서 AND, OR, NOT의 논리 연산자를 사용한 검색이 가능하게 하였다. 또한 검색의 복잡도를 \(O(N \cdot F)\) (\(N\)은 컴포넌트 개수, \(F\)는 패턴 값의 개수)로 \(O(N)\)으로 향상시킨다.

이 논문의 구성은 다음과 같다. 2장에서 컴포넌트에 관한 정보를 얻는 방법의 문제점과 컴포넌트 저장소에 있는 컴포넌트의 검색 방법을 제시한다. 3장에서 확장된 소프트웨어 컴포넌트 서술자를 제시하며 서술자에서 추출한 패턴을 정의한다. 4장에서 논리 연산자를 사용한 새로운 패턴 검색 방법과, 검색의 복잡도를 줄이는 방법을 제안한다. 마지막으로 5장에서 결론과 향후 연구 방향을 기술한다.

2. 컴포넌트의 정보 얻기와 검색 방법

2.1 컴포넌트에 관한 정보 얻기

대표적인 4종의 컴포넌트에 대하여 컴포넌트에 관한 정보를 얻는 방법을 기술하고 비교하며 문제점을 제시한다.

2.1.1 Enterprise Java Beans

Enterprise Java Beans[EJB][13]의 엔터프라이즈 범에 관한 정보는 인터페이스와 XML로 표시된 배치 서술자(deployment descriptor)의 2가지로 얻을 수 있다.

컴포넌트와 배치 서술자는 JAR 파일에 패키지와인 인터페이스 정보는 패키지되어 있지 않으며 또한 인터페이스에 대한 정보도 없으므로 저의 반사(reflection) API를 사용하여야 한다.

2.1.2 JavaBeans

JavaBeans[14]의 JAR 파일 내에 있는 목록(manifest) 파일은 속성과 같은 형식으로 표현되며 자바 빌드 메인 클래스, 다른 빌드 세부 내용, 설명 시점에 필요한 반인지를 알려 준다.

그러나 목록 파일은 컴포넌트에 대한 일반적인 정보를 제공하지 않으며 또한 인터페이스 정보를 얻기 위해 자바 빌드에 있는 내성(introspection) 기능을 사용하여야 한다.

2.1.3 CORBA 객체

CORBA[15]는 프로그래밍 언어 독립된 Interface Definition Language(IDL)을 사용하여 CORBA 객체의 서비스를 표현한다. IDL로 정의된 CORBA의 인터페이스는 별도의 장소인 인터페이스 저장소에 보관된다.

그런데 문제는 CORBA 객체의 인터페이스가 인터페이스 저장소에 보관되어 있지 않거나 또는 저장이 되어 있어도 인터페이스 저장소 자체에 접근할 수 없어서 필요한 정보를 얻을 수 없는 경우가 많아[5].

2.1.4 CORBA 컴포넌트

CORBA 3에서 소개된 CORBA 컴포넌트[6-8]의 인터페이스는 앞에서 서술한 CORBA 객체와 동일하게 인터페이스 저장소에 저장된다. CORBA 컴포넌트에는 XML 이름을 사용한 서술자로 소프트웨어 컴포넌트 서술자, CORBA 컴포넌트 서술자, 컴포넌트 어 velit리 서술자가 있다.

소프트웨어 컴포넌트 서술자는 컴포넌트에 대한 일반적인 정보를 가지고 있으나 일부 정보의 보완이 필요하여 이에 대한 자세한 내용은 3.2절에서 기술한다.

2.1.5 다른 프로그래밍

프로그래밍 형태가 아닌 프로그래머들은 프로그래밍에 대한 설명이나 소스 코드에 있는 주석과 같이 프로그래밍에 대한 자료를 사용할 수 있는데 많은 경우 이런 자료들이 상세한 정보를 제공하지 못한다[9].

이상에서 살펴본 것처럼 컴포넌트에 대한 정보를 얻는 방법이 컴포넌트의 종류에 따라 다르며(표 1) 참조) 정보를 얻을 수 있는 기본적인 기능은 갖추어져 있지만 실제로 는 정보를 얻는 것이 불가능한 경우도 있다. 이에 대한 가정은 대안은 XML과 같은 표준적인 도구를 사용하여 컴포넌트에 대한 정보를 표현하고 이를 컴포넌트와 함께 제공하여 유동하고 배치하는 것이다.

<table>
<thead>
<tr>
<th>항목</th>
<th>CORBA 객체</th>
<th>CORBA 컴포넌트</th>
<th>EJB</th>
<th>JavaBeans</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAR 파일</td>
<td>×</td>
<td>○</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>IDL 저장소</td>
<td>○</td>
<td>○</td>
<td>×</td>
<td>○</td>
</tr>
<tr>
<td>내장 기능</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>○</td>
</tr>
<tr>
<td>반사 기능</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>인터페이스에 관한 정보 얻기</td>
<td>IDL 저장소</td>
<td>○</td>
<td>반사</td>
<td>내장, 반사</td>
</tr>
<tr>
<td>XML 사용</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

2.2 컴포넌트 저장소 검색 방법

컴포넌트를 표현하고 검색하는 방식으로 제안된 대표적인
4가지의 인공 지능 방식([16, 17], 도서관학 방식([9-12, 18, 19], 형식 방식 방식([20-21]), 브라우저 방식([22, 23])가 이 점에서 본 논문과 관련된 도서관학 방식과 패치 방식을 서술한다.

2.2.1 도서관학 방식(library science approach)
도서관학 방식은 세 가지로 구분된다. 첫 번째로 나열 방법(enumerative method)은 대상 영역을 상호 독립적인 사례 구조로 분할한다. 대표적인 예가 뉴욕의 전쟁 시스템(18)이 다. 이 방법의 장점은 고도의 구조적인 표현으로 인해 사용자가 표현된 대상들간의 관계를 쉽게 이해한다는 것이고 단점은 이러한 제중 구조를 만드기 어렵다는 것이다.

두 번째의 패치 방법은 다음에서 별도로 설명한다. 세 번째 자율 텍스트 인덱스 방식([19])는 자주 사용하는 단어를 추출하여 컴퓨터를 인식시킨다. 컴퓨터의 설명이 주어지면 자율 텍스트 인덱스 방법은 설명에 나타난 단어의 빈도가 높은 컴퓨터로 찾아낸다. 이 방법은 프로그램 상에서 주석의 가용성 및 정확성의 문제로 인해 컴퓨터를 정확하게 표현하기 어렵다.

2.2.2 패치 방식
도서관학 방식의 하나인 패치 방식([9-12])는 핵심을 분류하는 데 핵심적인 방법을 사용하여 나열 방식에 비해 핵심의 확장과 관리는 더욱 정확도와 표현력이 좋다.

패치를 사용한 방법에서는 서로 다른 종류의 정보를 표현하는데 각각 한 개의 패치를 사용한다. 여러 개의 패치로 구성된 패치 목록(그림 1)에서 F_1, F_2, ..., F_n이 컴퓨터를 표현한다. 한 개의 패치는 동일한 종류의 정보를 표현하는 다양한 형태를 표현한다. 컴퓨터는 한 개의 패치에 대해 여러 개의 패치 값을 가질 수 있다. (그림 1)의 핵심 옵션 σ와 그것을 패치 값으로 표현한 것을 보여 주고 있다.

(그림 1) 패치에 의한 컴퓨터 표현

3. 소프트웨어 컴퓨터 시술자의 확장

3.1 소프트웨어 컴퓨터 시술자(software component descriptor)
소프트웨어 패키지는 시술자의 단위로 소프트웨어를 설명하는데 필요한 파일의 집합으로 구성된다. 시술자는 패키지의 목록을 표현하고 패키지에 있는 여러 파일에 대한 정보를 가지고 있다.

개방형 소프트웨어 시술자(open software description, OSD)는 Corba, 싱글린이 포함된 W3C에 제안한 것으로 XML과 기여하여 소프트웨어 패키지와 패키지간의 통신을 시술하는 어휘(vocabulary)를 제공한다.

컴퓨터 패키지는 소프트웨어 패키지의 한 종류이다. CORBA 3의 소프트웨어 패키지는 OSD를 확장한 소프트웨어 컴퓨터 시술자(SCD)로 사용하여 컴퓨터의 정보를 기술한다. SCD, 그리고 다음 제2에 서술할 확장된 SCD의 XML 요소를 표 2에 정리하였다.

<table>
<thead>
<tr>
<th>표 2</th>
<th>컴퓨터 시술자의 요소들</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCD의 요소</td>
<td>SCD의 요소</td>
</tr>
<tr>
<td>softpkg, implementation</td>
<td>OSD의 요소(softpkg, implementation, dependency, title, license, os, version)</td>
</tr>
<tr>
<td>dependency, title, license, os, version, processor, abstract, codebase, impvity, language, vm, memsize, directory</td>
<td>author, code, compiler, company, descriptor, filearchive, idl, humanlanguage, link, localfile, plugtype, programminglanguage, propertyfile, repository, resource, runtime, threadsafety, webpage</td>
</tr>
</tbody>
</table>

3.2 소프트웨어 컴퓨터 시술자의 확장
기본적으로 소프트웨어 컴퓨터 시술자는 여러 정보를 포함하고 있지만 이것은 컴퓨터를 설명하지만 또는 컴퓨터를 설명하는 컴퓨터의 정보에서 필요한 중요한 정보를 제공하지 못하므로 이 점
에서 이에 대한 확장을 한다. 확장되는 내용은 다음과 같다.

(1) 인터페이스: 컴포넌트의 인터페이스는 컴포넌트가 제공하는 서비스와 서비스를 사용하는 방법을 알려주는 기본적인 정보로 2.1절에서 지시된 것처럼 이에 대한 정보를 얻는 것이 쉬운 일이 아니다. 따라서 인터페이스를 공통으로 형식으로 표시하여 자료 표현이 쉽고 프로그램에서 처리가 용이하도록 XML 형식으로 표시한다. 사용자는 이 정보를 사용하여 컴포넌트를 쉽게 발견할 수 있다. 인터페이스에 포함되는 중요한 정보는 다음과 같다.

- 메소드 이름(method name) : 인터페이스에 정의된 메소드의 이름이다.
- 메개변수 리스트(parameter list) : 메소드에 정의된 메개변수 리스트를 표시한다.
- 반환형(return type) : 반환형은 오픈레이션이 끝난 후에 호출자가 얻을 값을 표시한다.
- 예외사항 리스트(exception list) : 메소드에서 발생 가능한 모든 예외사항들을 표시한다.

(2) 업무영역(domain) : 업무영역은 컴포넌트의 속한 업무를 지칭한다. 가능한 값으로는 "production", "public", "finance"등을 들 수 있다. 이 자료는 다른 업무 영역에 있는 비슷한 컴포넌트를 구별하여 사용된다.

(3) 컴포넌트 기능 : 컴포넌트가 제공하는 인터페이스에 그 컴포넌트가 제공하는 메소드와 그 메소드를 사용하는 방법을 표현하지만 컴포넌트가 하는 일을 직접적으로 표현하지 않는다.


<input, characters, buffer>, (substitute, tabs, file), (search, root, B-tree), (compress, lines, file)

*표 3*의 DTD 정의는 이 논문에서 SCD를 새로 추가한 요소인 idl-spec, domain, comp-function을 표시하고 있다. idl-spec은 CORBA의 인터페이스를 표현하는 IDL을 기본으로 하여 만든 것이며 그 종의 일부를 표시하였다. 이 정의는 CORBA 컴포넌트와 CORBA 이외의 EJB와 JavaBeans를 표현할 수 있는 일반성을 지니고 있다.

idl-spec 정의는 CORBA와 Java의 인터페이스에 없는 내용(예를 들면 메소드의 방향, 메개변수의 방향 등)을 포함하고 있는데 심화적인 사항으로 정의되어 있으므로 Java의 인터페이스를 표현하는데 문제가 없다. Java의 인터페이스에 없고 CORBA에 없는 package, import, 인터페이스와 메소드의 접근수정자(access modifier)들을 선택 사항으로 정의하였다.

*표 4*에 CORBA의 IDL 예가 있으며 *표 5*에서 이를 XML 형식으로 표시하였다. 또한 *표 6*에 EJB의 인터페이스의 예가 있으며 *표 7*에서 이를 XML 형식으로 표시하였다.

*표 3* 확정된 소프트웨어 컴포넌트 서블트를 위한 DTD

```
<!ELEMENT escd (idl-spec | domain | comp-function)>  
<!ELEMENT idl-spec (type-def | const-def) | except-def | interface-def | module-def | component-def | home-def | package-def | import-def)>  
<!ELEMENT interface-def (interface | forward-interface)>  
<!ELEMENT forward-interface (interface-name)>  
<!ELEMENT interface (super-interface | type-def | const-def | except-def | attr-def | method-def)>  
<!ATTLIST interface name CDATA #REQUIRED>  
<!ATTLIST interface access-modifier (public | package) #IMPLIED>  
<!ELEMENT super-interface (#PCDATA)>  
<!ELEMENT method-def (method-direction?, return-type, parameter-def*, exception-def*)>  
<!ATTLIST method-def name CDATA #REQUIRED>  
<!ATTLIST method-def direction EMPTY >  
<!ATTLIST method-def direction (oneway | REQUIRED >  
<!ATTLIST return-type (#PCDATA)>  
<!ELEMENT parameter-def (parameter-direction?, parameter-type)>  
<!ATTLIST parameter-def name CDATA #REQUIRED>  
<!ATTLIST parameter-definition EMPTY >  
<!ATTLIST parameter-definition value (in | out | inout) #REQUIRED>  
<!ELEMENT parameter-type (#PCDATA)>  
<!ELEMENT exception-def (exception-type, exception-data)>  
<!ATTLIST exception-def name CDATA #REQUIRED>  
<!ELEMENT exception-type (#PCDATA)>  
<!ELEMENT exception-data (#PCDATA)>  
<!ATTLIST attr-def (attr-read, attr-type) >  
<!ATTLIST attr-def name CDATA #REQUIRED>  
<!ATTLIST attr-type (#PCDATA)>  
<!ATTLIST attr-read EMPTY >  
<!ATTLIST attr-read value (readonly) >  
<!ELEMENT package-def EMPTY >  
<!ATTLIST package-def name CDATA #REQUIRED>  
<!ELEMENT import-def EMPTY >  
<!ATTLIST import-def name CDATA #REQUIRED>  
<!ELEMENT domain (#PCDATA)>  
<!ELEMENT comp-function (function, object, medium)>  
<!ELEMENT function (#PCDATA)>  
<!ELEMENT object (#PCDATA)>  
<!ELEMENT medium (#PCDATA)>  
```
자료가 있다고 가정한다. 그러나 2.1절에서 살펴보았듯이 실제로는 코 viewController 저장소에서 사용할 컴포넌트에 대한 정보를 얻는 방법은 쉽지 않으며 컴포넌트 및 프로그램의 종류에 따라 다르다. 따라서 이 논문에서는 다른 논문들에서 다루지 않았던 코생활을 표현하는 방법으로 XML을 표시된 확장된 SCD를 제안한다.

확장된 SCD는 코생활의 이해와 EJB와 같은 다른 코생활을 하지 못한 코생활자에 사용할 수 있다. Java의 인터페이스 이론 논문에서 제안한 확장된 SCD를 사용하여 표현할 수 있음을 3.2절에서 보였다. 인터페이스를 사용하지 않는 코생활자 예시의 경우는 외부로 사용할 수 있는 메소드에 대하여 그 메소드의 이름, 반환형, 매개 변수 등을 확장된 SCD를 사용하여 표현하고 다른 메소드에 대한 일반 정보도 확장된 SCD를 사용하여 표현이 가능하다.

일반적으로 코생활은 ZIP의 형태로 압축하여 빌드가 된다. 확장된 SCD의 확장자는 scd로 하여 ZIP 파일의 메타데이터로 저장하여 유용함으로 코생활에 관한 정보를 쉽게 얻을 수 있다.

3.3 확장된 소프트웨어 코생활 서술자 유용성
기존의 코생활 저장소의 연구는 이미 코생활에 대한

3.3 확장된 소프트웨어 코생활 서술자 유용성
기존의 코생활 저장소의 연구는 이미 코생활에 대한

3.4 추출한 패턴
소프트웨어 코생활 서술자는 나열 방법[18] 등에서 요구하는 별도의 문구나 조작이 없이 됨하게 패턴으로 전달될 수 있다. 즉 ELEMENT에 표현된 내용과 ATTRIBUTE에 있는 내용을 한 개의 패턴으로 만들 수 있다. 예를 들면 XML로 표시된 <function>delete</function>에서 function이 한 개의 패턴이 되고 delete가 패턴의 일부가 된다.

패턴 목록은 코생활자 동작을 하는 일과 그 일에 대한 방법을 감각하여 표현할 수 있어야 하며 코생활의 본체와 코생활자에 필요한 충분한 정보를 가지고 있으며 또한 단순 명령어 하나로 하여 한다[9], 이런 원칙에 근거하여 3.1절과 3.2절에서 서술한 내용에서 추출한 패턴을 아래에 서술하였다.

(1) 이름(name) : 코생활의 이름을 표시한다.
(2) 버전(version) : 컴포넌트의 버전을 표시한다. 버전을 표시하는 형태는 중요한 버전과 부수적인 버전을 숫자와 위표로 표시한다. 예로 들면 1.0.0,0이다.
(3) 패키지 타입(pkgtype) : 컴포넌트의 종류를 나타낸다. SCD에 “CORBA Component”와 “CORBA Interface Impl”이 CORBA 컴포넌트와 CORBA 객체를 위해 예약되어 지정되어 있다. 자바를 위해서 “JavaBeans”와 “EJB”를, 일반 프로그램을 위해 “others”를 추가한다.
(4) 인터페이스 : 3.2절에서 정의한 메소드 이름과 예의 사항을 사용한다.
(5) 업무영역(domain) : 3.2절에서 정의한 업무 영역을 사용한다.
(6) 컴포넌트 기능(function) : 3.2절에서 정의한 function, object, medium을 사용한다.
(7) 코드의 위치(code) : 컴포넌트가 동일 지역에 있는지 원격지에 있는지 구별하여 준다. 컴포넌트 저장소가 중앙집중이 아닌 분산화 될 수 있으며 또한 인터넷상에 있는 컴포넌트에 대한 정보를 가질 수 있으므로 필요한 패킷이다.
(8) 컴파일러(compiler) : 코드를 생성한 컴파일러를 정지한다.
(9) 종속성(dependency) : 수행할 때 환경적인 종속성을 표현한다. 예를 들어 ORB를 정지할 수 있다.
(10) 프로그래밍 언어(programming language) : 컴포넌트를 구현한 프로그래밍 언어를 표시한다.
(11) 운영체제(os) : 컴포넌트가 수행되는 운영체제를 표시한다.
(12) 프로세서(processor) : 컴포넌트가 수행될 수 있는 프로세서의 종류를 표시한다.
(13) 런타임(runtime) : 컴포넌트가 요구하는 런타임을 정지한다. 예로 Java VM과 같은 것이다.

4. 컴포넌트 저장소의 검색

이 장에서 논리 연산자를 사용한 새로운 패킷 검색 방법을 제안한다. 또한 패킷 값의 척자들을 생성하고 부정 검색을 위한 가중치 신경 접속 행렬을 구하는 알고리즘을 제안하여 검색의 복잡도를 향상할 것을 보인다.

4.1 신경 접속 행렬

이 장에서 사용하는 변수들은 표 9에 정의되어 있다.

<table>
<thead>
<tr>
<th>변수</th>
<th>정의</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>저장소에 있는 컴포넌트의 개수</td>
</tr>
<tr>
<td>n_i</td>
<td>패킷 값 i를 가지고 있는 컴포넌트의 수</td>
</tr>
<tr>
<td>F</td>
<td>다른 패킷 값의 개수</td>
</tr>
</tbody>
</table>

정의 1 : 신경 접속 행렬 W의 원소\(w_{ij}\)

\[w_{ij} = 1 \text{ if component } j \text{ has facet value } i \]
\[w_{ij} = 0 \text{ otherwise} \]

정의 2 : 사용자 검색 토큰 Q의 원소\(q_i\),\(1 \leq i \leq F\)

\[q_i = 1 \text{ if a component with facet value } i \text{ is needed} \]
\[q_i = 0 \text{ otherwise} \]

정의 3 : 검색 출력 벡터\(O\)

\[O = Q \cdot W\]

\(Q \cdot W\)를 계산하면 N개의 원소를 가진 출력 벡터\(O\)가 만들어지는데,\(O\) 값이 컴포넌트\(i\)의 검색 만족도가 된다.

예 1. 신경 접속 행렬

컴포넌트의 기능(function)을 표현하는 패킷\(A, A\)에 속한 3개의 패킷 값과 4개의 컴포넌트가 다음과 같다.\(A = \text{create, delete, substitute} \),\(C1 = \text{create} \),\(C2 = \text{create, delete, substitute} \),\(C3 = \text{delete} \),\(C4 = \text{substitute} \), 이 때 신경 접속 행렬\(W\)는 아래와 같이 정의된다.

\[W = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \]

사용자가 “delete”라는 패킷 값을 갖는 컴포넌트를 찾는다고 하자. 그러면 검색\(Q\)이 이에 대한 출력 벡터\(O\) 다음과 같다.

\[Q = \{0, 1, 0\} \rightarrow O = Q \cdot W = \{0, 1, 1, 0\} \]

결과는 컴퓨터 2와 3이 선택된다.

여러 컴퓨터가 동일한 패킷 값을 가질 수 있으며 또한 한 컴퓨터가 여러 개의 다른 패킷 값을 가질 수 있다. 이 때 컴퓨터가 가지는 패킷 값의 희소성과 컴퓨터가 검색하는 패킷 값에 대한 만족도를 정확하게 계산하기 위해 가중치 함수를 사용한 다음과 같은 행렬을 제안되었다[11].

정의 4 : 가중치 신경 접속 행렬

가중치 신경 접속 행렬의 원소\(w'_{ij}\)는 다음과 같이 정의한다.

\[w'_{ij} = \frac{\log(s(N/n_i))}{\sum_{k=1}^{F} w_{ik} \log(s(N/n_k))} \text{ s.t. } \sum_{i=1}^{F} w'_{ij} = 1\]
에 2. 가중치 신경 접속 행렬
에 1의 행렬 $W$에 가중치 합수를 적용하여 아래와 같은 가중치 신경 접속 행렬 $W'$를 구한다.

$$W' = \begin{bmatrix} 1 & 1/3 & 0 & 0 \\ 0 & 1/3 & 1 & 0 \\ 0 & 1/3 & 0 & 1 \end{bmatrix}$$

42 논리 연산자를 이용한 검색
앞 절의 가중치 합수는 한 개의 패시 값을 대해서는 작동하지만 두 개 이상이 되면 문제가 될 수 있다. 이에 대한 예를 살펴보면 다음과 같다.

예 3. 논리 연산기를 고려하지 않은 검색
절이어 $Q = \{0, 1, 1\}$에 대한 출력 벡터는 다음과 같다.

$$O = Q \cdot W' = \{0, 2, 1, 1\}.$$  

24.2.3 논리분산 검색
논리분산 검색의 출력 벡터는 다음과 같이 정의한다.

정의 6: 논리분산 검색 $\text{AND}$의 $i$번째 원소 $o_{AND}$,

$$o_{AND} = h_i \cdot o_{OR}$$

즉 $o_{OR}$와 $i$번째 원소와 히트율 벡터의 $i$번째 원소의 곱으로 구한다.

예 5. 논리분산 검색
절이어 $Q = \{0, 1, 1\}$에 대하여 히트율 벡터와 최종 검색 결과는 아래와 같다.

$$H = \{0, 1, 1, 1, 1\}.$$  

정의 7: 논리분산 검색의 출력 벡터 $O_{AND}$의 $i$번째 원소 $o_{AND}$,

$$o_{AND} = h_i \cdot o_{OR}$$

결과에서 보듯이, $O_{AND}$가 2개의 패시 값을 동시에 만족하므로 한 개의 패시 값만 만족하는 $O_{AND}$가 3개 4보다 만족도가 높게 나온다.

4.2.2 논리분산 검색
논리분산 검색의 출력 벡터를 다음과 같이 정의한다.

정의 8: 논리분산 검색의 출력 벡터 $O_{AND}$,$O_{OR}$,$O_{AND}$, $O_{OR}$의 $i$번째 원소 $o_{AND}$, $o_{OR}$, $o_{AND}$와 $o_{OR}$의 곱으로 구성한다.

$$H = \{0, 1, 1/2, 1/2\}.$$  

정의 9: 논리분산 검색의 출력 벡터 $O_{AND}$,$O_{OR}$,$O_{AND}$, $O_{OR}$의 $i$번째 원소 $o_{AND}$, $o_{OR}$, $o_{AND}$와 $o_{OR}$의 곱으로 구성한다.

$$\bar{k}_{w_i} = 1 - w_i, \quad \text{if } i = k$$

정의 9: 논리분산 검색의 출력 벡터 $O_{AND}$,$O_{OR}$,$O_{AND}$, $O_{OR}$의 $i$번째 원소 $o_{AND}$, $o_{OR}$, $o_{AND}$와 $o_{OR}$의 곱으로 구성한다.

$$\bar{k}_{w_i} = \bar{k}_{w_i} - \frac{\log_j(N/n_t)}{\sum_{i=1}^{n_t} k_{w_i} \log_j(N/n_t)}$$
예 6. 논리 부정 검색

정의

정의어 $Q = \{0, 1, 0\}$가 있다. $1$은 논리 부정 검색의 음

미한다. $k$ 가 2이므로 $W$에서 $\overline{2W}$ 와 $\overline{2W'}$를 다음과 같이

 얻는다.

\[
\overline{2W} = \begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1
\end{bmatrix}
\overline{2W'} = \begin{bmatrix}
1/2 & 1/2 & 0 & 0 \\
1/2 & 0 & 0 & 1 \\
0 & 1/2 & 0 & 1/2
\end{bmatrix}
\]

부정 검색 결과는 다음과 같다.

\[
O_{\overline{2W}} = Q \cdot 2W = \frac{1}{2, 0, 0, 1/2}
\]

논리부정 검색을 하기 위해 가중치 신경 접속 영향 $\overline{kW'}$ 전체를 새로운 만드는 것은 복잡도가 $O(N \cdot F)$이다. 또한 한 개의 원소인 $\overline{kW'}$를 구하는 복잡도는 $O(F)$이다. 복잡도를

향상시키기 위해 다음과 같은 벡터 $S$를 정의한다.

정의 10. 가중치 할 벡터 $S$

$S$의 원소 $s_j$는 다음과 같이 정의한다.

\[
s_j = \sum_{i=1}^{F} w_{ij} \cdot \log \left( \frac{N}{N-n_i} \right) \quad (1 \leq j \leq N)
\]

이와 같이 정의된 $S$를 미리 계산하여 저장하고 있으면

패치 값 $k$를 부정한 신경 접속 영향의 원소 $\overline{kW'}$는 다음과 같이 구할 수 있다.

<table>
<thead>
<tr>
<th>신경 접속 영향의 원소 $\overline{kW'}$를 구하는 알고리즘</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{ij}$ = 0</td>
</tr>
<tr>
<td>$i = k$</td>
</tr>
<tr>
<td>$\overline{kW'}<em>{ij} = w</em>{ij} \cdot s_j$</td>
</tr>
<tr>
<td>$i \neq k$</td>
</tr>
<tr>
<td>$\overline{kW'}<em>{ij} = \frac{w</em>{ij} \cdot s_j}{s_j + \log(N(N-n_i))}$</td>
</tr>
<tr>
<td>$\overline{kW'}<em>{ij} = \frac{w</em>{ij} \cdot s_j}{s_j + \log(N(N-n_i))}$</td>
</tr>
<tr>
<td>에게 $i = k$</td>
</tr>
<tr>
<td>$\overline{kW'}<em>{ij} = w</em>{ij} \cdot s_j$</td>
</tr>
<tr>
<td>$\overline{kW'}<em>{ij} = \frac{w</em>{ij} \cdot s_j}{s_j - \log(Nn_i)}$</td>
</tr>
</tbody>
</table>

위의 알고리즘을 사용하면 $\overline{kW'}$를 구하는 복잡도가 $O(F)$

에서 $O(1)$으로 환상화한다.

예 7. 논리부정 검색을 위한 신경 접속 영향

예 1의 $W$에 대한 가중치 할 벡터 $S$는 다음과 같다.

$S = \{1, 3, 1, 1\}$

정의어 $Q = \{0, 1, 0\}$에 대한 $\overline{2W'}$의 원소를 위의 알고리즘을 사용하여 아래와 같이 구할 수 있으며 예 6의 결과와 동일한 값을 얻을 수 있다.

\[
\overline{2W'}_{11} = 0, \quad \overline{2W'}_{12} = 0
\]
\[
\overline{2W'}_{21} = \frac{1}{2}, \quad \overline{2W'}_{22} = \frac{1}{2}, \quad \overline{2W'}_{23} = 0
\]
\[
\overline{2W'}_{31} = \frac{1}{2}, \quad \overline{2W'}_{32} = \frac{1}{2}
\]

이와 같이 논리 연산자가 동시에 사용될 때의 예를 살펴보자.

예 8. NOT과 OR의 예

$Q_{OR} = \{1, 1, 0\}$이면 검색 결과는 다음과 같다.

$O_{OR} = Q \cdot 2W' = \{1, 1/2, 0, 1/2\}

예 9. NOT과 AND의 예

$Q_{AND} = \{1, 1, 0\}$이면 허트를 입력해와 최종 검색 결과는 다음과 같다.

\[
H = \{1, 1/2, 0, 1/2\}
\]

$O_{\overline{2W}} = \{1 \cdot 1, 1, 1/2, 0, 1, 0, 1/2 \cdot 1/4, 0, 1/4\}$

4.3 신경 접속 영향의 제산의 복잡도

패치 값 전체의 계산은 상당히 많으나 일반적으로 한 개의 검색에 사용하는 패치 값은 매우 적다. 따라서 영향 전체를 계산한 필요없이 행렬에서 검색하는 패치 값의 행 벡터만 선택하여 연산한다. 즉, 정의어 $Q$에서 $0$에 해당하는 행렬의 행은 무시하고 $1$이나 $\overline{1}$로 표시한 행만 선택한다.

$1$로 표시된 패치 값에 대해서는 $\overline{kW'}$을 구하는 알고리즘을 재활용한 방법을 사용하여 선택된 행에 대해서만 $\overline{kW'}$을 구하면 되므로 논리부정 검색을 위한 가중치 신경 접속 영향을 구하는 복잡도는 $O(N)$이다. 따라서 영향 연산을 포함한 전체적인 복잡도는 $O(N \cdot F)$[11]으로 향상되었다.

4.4 동의어 사전

한 사전을 표현하는데 동의어를 사용하여 다르게 표현할 수 있다. 따라서 어떤 패치 값으로 정의된 침포먼트가 없더라도 그 패치 값의 동의어로 정의된 침포먼트를 찾을 수 있다. 이를 지원하기 위한 것이 동의어 사전이다. 동의어를 저장하는 여러 시스템들이 있는데 이 논문에서는 2중 구조의 동의어 사전을 제안하여 효율적인 검색을 지원한다.

(1) 패치 값 사전: 이 사전은 현재 침포먼트 저장소에서 저장되어 있는 침포먼트가 소유하고 있는 패치 값을 가지 고 있다. 또한 각 패치 값에 대한 인덱스를 가지고 있다.
4.5 컴퓨포넌트 검색 과정

이 절에서 종합적으로 컴퓨포넌트의 검색 과정을 서술한다. (그림 2)에 컴퓨포넌트 시스템의 구조와 검색 과정이 표시되어 있다.

(1) 사용자가 원하는 패킷 값을 입력하여 해당되는 컴퓨포넌트의 검색을 요청한다.
(2) 검색 서버는 고객의 요청을 받아 해석하고 그 내용을 첨자 생성자에게 보낸다.
(3) 첨자 생성자는 고객이 입력한 패킷 값을 패킷 값 사전에 보낸다.
(4) 사전에 패킷 값이 존재하면 첨자 (r)를 구하여 첨자 생성자에게 보낸다.
(5) 만약 패킷 값 사전에 패킷 값이 없으면 동의어 사전에서 해당하는 패킷 값이 있는지 확인한다.
(6) 만약 동의어 항목에 해당하는 패킷 값 사전이 존재하지 않는다면 동의어 항목에 해당하는 패킷 값의 첨자 (k)를 구하여 첨자와 적합도 (r)값을 보낸다.
(7) 사전에서 얻은 첨자와 적합도를 출력버티 생성자에게 보낸다.
(8) 넘겨받은 각 첨자 r에 대해 신경 접속 형렬의 r 번째 행 $W_r$를 추출한다. 동의어 사전에서 첨자 k에 대한 행은 신경 접속 형렬의 k 번째 행을 추출하여 $W_k = r, W_k$를 구한다.
(9) 컴퓨포넌트 생성자가 추출된 모든 행을 사용하여 컴퓨포넌트의 출력 벡터를 구한다. 출력 벡터의 유전은 42에서 보인 것처럼 논리 연산자에 따라 계산 결과에 달려있다.

5. 결론 및 향후 연구 방향

본 논문에서 인터페이스와 컴퓨포넌트의 기능 등을 XML 어휘로 표현하였고 이를 CORBA 3의 소프트웨어 컴퓨포넌트 서무자에 포함하여 서무자를 확장하였다. 이 확장된 서무자는 CORBA 컴퓨포넌트, CORBA 객체, EJB, JavaBeans 및 일반 프로그램 등에 적용할 수 있다. 확장된 소프트웨어 컴퓨포넌트 서무자에서 유용한 정보를 추출하여 이를 패킷으로 만들고 컴퓨포넌트 저장소에 있는 컴퓨포넌트를 검색하는데 사용하였다. 또한 확장된 소프트웨어 컴퓨포넌트 서무자와 함께 편리하고 유효하고 배치한 정보를 제공하였다.

이렇게 함으로써 특정한 프로그래밍 과정이나 인터페이스 저장소에 의존하던 기존의 방법을 풀어서 어떤 종류의 컴퓨포넌트에 대해서도 표준적이고 신속한 방법으로 인터페이스에 관한 정보를 얻을 수 있다. 또한 컴퓨포넌트에 관한 일반적인 정보나 컴퓨포넌트 저장소의 검색에 필요한 정보도 확장된 소프트웨어 컴퓨포넌트 서무자를 통하여 용이하게 얻을 수 있다. 컴퓨포넌트 개발자 및 사용자들은 확장된 소프트웨어 컴퓨포넌트 서무자를 표준화하여 사용하면 컴퓨포넌트 검색뿐만 아니라 컴퓨포넌트의 유통 및 배치에도 많은 혜택이 될 것이다.

본 논문에서 제안한 패킷 검색 방법은 기존의 패킷 방법에서 지원되지 않은 논리 연산자 사용된 검색이 가능하게 하였다. 또한 논리 부정 검색을 위한 가중치 설정 검색 행렬을 계산하는 알고리즘을 개발하고 검색하는 패킷 값의 첨자를 사용하여 검색 시간의 복잡도를 향상하였다.

향후 연구 과제로 이 논문에서 제안한 시스템을 구현하고 기존의 다른 시스템과 비교·평가한 실험 결과를 추가하는 입니다. 또한 컴퓨포넌트 저장소의 검색에서 이 논문에서 제안한 논리 연산자를 사용한 검색뿐 아니라 CORBA의 트레이닝 객체 서비스에서 요구하는 다양한 첨자 지원되는 방법이 요구된다.

참고 문헌


