확장된 소프트웨어 컴포넌트 서술자에 기초한 컴포넌트 저장소의 검색

금 영 욱[†] · 박 병 섭[††]

요 약

컴포넌트 저장소의 효율적인 검색이 컴포넌트 재사용에 매우 중요하다. 컴포넌트 저장소에 보관할 컴포넌트에 대한 정보를 얻는데 일반적으로 많은 시간과 노력이 필요하다. CORBA 3의 컴포넌트 소프트웨어 서술자는 XML을 사용하여 일반적인 컴포넌트의 특성을 서술한다. 본 논문에서 CORBA 3의 소프트웨어 컴포넌트 서술자를 확장하며 이를 사용하여 컴포넌트 저장소의 검색에 필요한 정보를 얻는다. 패싯에 기초한 새로운 검색 방법을 제안하여 기존의 패싯 방법에서 지원하지 않았던 논리 연산자를 사용한 검색이 가능하며 또한 검색의 복잡도가 향상된다.

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), 패싯(facet), 소프트웨어 컴포넌트(software component), 소프트웨어 컴포넌트 서술자(software component descriptor), 검색 방법(retrieval method)

1. 서 론

소프트웨어 개발의 복잡성과 가격이 대폭적으로 증가하면서 80년대 중반 이후 효율적인 소프트웨어 재사용이 점점 중요하게 되었다. 소프트웨어 재사용을 위한 현재까지 알려진 최선의 방법은 컴포넌트에 기반한 소프트웨어를 만드는 것이다. 컴포넌트에 기반한 소프트웨어 제작을 위해서 해결해야 하는 중요한 문제 중의 하나는 점점 크기가 커지는 소프트웨어 저장소에서 원하는 소프트웨어 컴포넌트를 효율적으로 검색하는 일이다[1-3].

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

컴포넌트 저장소에 있는 컴포넌트를 효율적으로 검색하려면 먼저 컴포넌트에 관한 올바른 정보를 가지고 있어야 한다. 그러나 컴포넌트에 관한 정보(특별히 인터페이스에 관한 정보)를 수집하는데 많은 어려움이 있으며 컴포넌트에 관한 정보를 얻는 방법이 컴포넌트의 종류에 따라 다르며 복잡한 프로그래밍을 통해 가능하다[5]. CORBA 3[6, 7]의 소프트웨어 컴포넌트 서술자(Software Component Descriptor)[8]와 같이 컴포넌트를 표현하는 방법이 일부 존재하지만 컴포넌트 검색에 활용되고 있지 않다.

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

이 논문의 기여하는 바는 크게 세 가지이다. 첫 번째로 컴포넌트의 인터페이스와 다른 정보들을 XML 어휘로 표현

† 정 회 원 : 성결대학교 컴퓨터학부 교수
†† 정 회 원 : 우석대학교 컴퓨터교육과 교수
논문접수 : 2001년 1월 8일, 심사완료 : 2002년 3월 18일

하여 CORBA 3에서 제안된 소프트웨어 컴포넌트 서술자를 확장한다. 컴포넌트의 제작자가 컴포넌트와 확장된 소프트웨어 컴포넌트 서술자를 함께 패키징하여 유통시킴으로 컴포넌트 사용자가 컴포넌트에 대한 정보를 쉽게 얻을 수 있다. 확장된 서술자는 CORBA 컴포넌트 외에 다른 컴포넌트들과 일반 프로그램에도 사용 가능하다.

두 번째로 확장된 소프트웨어 컴포넌트 서술자에서 자료들을 추출하여 컴포넌트 검색에 이용한다. 컴포넌트를 검색하는 방법으로 패싯(facet)[9-11]을 사용하는데 확장된 소프트웨어 컴포넌트 서술자에서 추출한 자료들은 패싯으로 전환이 용이하다.

세 번째로 패싯 검색 방법으로 신경 접속 행렬[11]을 확장하여, 다른 패싯 검색[9-12]에서는 기본 검색만 제공하였지만 본 논문에서 AND, OR, NOT의 논리 연산자를 사용한 검색이 가능하게 하였다. 또한 검색의 복잡도를 $O(N \cdot F)$ (N은 컴포넌트 개수, F는 패싯 값의 개수)에서 $O(N)$으로 향상한다.

이 논문의 구성은 다음과 같다. 2장에서 컴포넌트에 관한 정보를 얻는 방법의 문제점과 컴포넌트 저장소에 있는 컴포넌트의 검색 방법을 기술한다. 3장에서 확장된 소프트웨어 컴포넌트 서술자를 제안하며 서술자에서 추출한 패싯을 정의한다. 4장에서 논리 연산자를 사용하는 새로운 패싯 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 4개의 컴포넌트에 대하여 컴포넌트에 관한 정보를 얻는 방법을 기술하고 비교하며 문제점을 제시한다.

2.1.1 Enterprise Java Beans

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

컴포넌트와 배치 서술자는 JAR 파일에 패키징되나 인터페이스 정보는 패키징되어 있지 않으며 또한 인터페이스를 위한 별도의 저장소가 없다. 따라서 EJB에서 인터페이스에 대한 정보를 얻으려면 저급의 반사(reflection) API를 사용하여야 한다.

2.1.2 JavaBeans

JavaBeans[14]의 JAR 파일 내에 있는 목록(manifest) 파일은 속성과 값의 쌍으로 표현되며 자바 빈의 메인 클래스, 다른 빈에 종속 여부, 설계 시점에만 필요한 빈인지 알려준다.

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

2.1.3 CORBA 객체

CORBA[15]는 프로그래밍 언어와 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 많다[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 어휘를 사용한 서술자로 소프트웨어 컴포넌트 서술자, CORBA 컴포넌트 서술자, 컴포넌트 어셈블리 서술자가 있다.

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

2.1.5 다른 프로그램

컴포넌트 형태가 아닌 프로그램들은 프로그램에 대한 설명서나 소스 코드에 있는 주석문이 그 프로그램에 대한 자료로 사용될 수 있는데 많은 경우 이런 자료들이 상세한 정보를 제공하지 못한다[9].

이상에서 살펴 본 것처럼 컴포넌트에 대한 정보를 얻는 방법이 컴포넌트의 종류에 따라 다르며(<표 1> 참조) 정보를 얻을 수 있는 기본적인 기능은 갖추어져 있지만 실제로는 정보를 얻는 것이 불가능한 경우도 있다. 이에 대한 가장 좋은 대안은 XML과 같은 표준적인 도구를 사용하여 컴포넌트에 대한 정보를 표현하고 이를 컴포넌트와 함께 패키징하여 유통하고 배치하는 것이다.

〈표 1〉 컴포넌트의 비교

항 목	CORBA 객체	CORBA 컴포넌트	EJB	JavaBeans
컴포넌트 서술자	×	○	×	×
IDL 저장소	○	○	×	×
내성 기능	×	×	×	○
반사 기능	×	×	○	○
인터페이스에 관한 정보 얻기	IDL 저장소	IDL 저장소	반사	내성, 반사
XML 사용	×	○	○	×

2.2 컴포넌트 저장소 검색 방법

컴포넌트를 표현하고 검색하는 방식으로 제안된 대표적인

4가지는 인공 지능 방식[16, 17], 도서관학 방식[9-12, 18, 19], 형식 명세 방식[20-21], 브라우징 방식[22, 23]이다. 이 절에서 본 논문과 관련된 도서관학 방식과 패싯 방식을 서술한다.

2.2.1 도서관학 방식(library science approach)

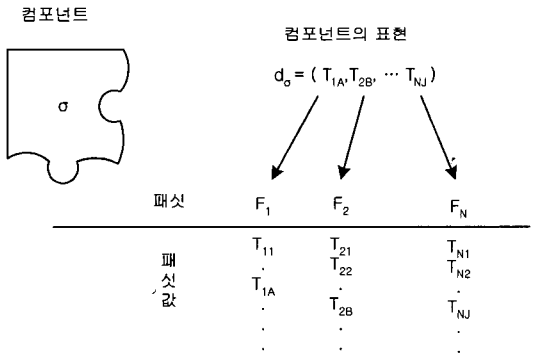
도서관학 방식은 세 가지로 구별된다. 첫 번째로 나열 방법(enumerative method)은 대상 영역을 상호 독립적인 계층 구조로 분할한다. 대표적인 예가 듀이의 십진 시스템[18]이다. 이 방법의 장점은 고도의 구조적인 표현으로 인해 사용자가 표현된 대상들간의 관계를 쉽게 이해한다는 것이고 단점은 이러한 계층 구조를 만들기 어렵다는 것이다.

두 번째의 패싯 방법은 다음절에서 별도로 설명한다. 세 번째 자유 텍스트 인덱스 방법[19]은 자주 사용하는 단어를 추출하여 컴포넌트를 인덱싱한다. 컴포넌트의 설명이 주어지면 자유 텍스트 인덱스 방법은 설명서에 나타난 단어의 빈도가 높은 컴포넌트를 찾아온다. 이 방법은 프로그램 상에서 주석의 가용성과 정확성의 문제로 인해 컴포넌트를 정확하게 표현하지 못한다.

2.2.2 패싯 방식

도서관학 방식의 하나인 패싯 방식[9-12]은 항목을 분류하는데 합성적인 방법을 사용하므로 나열 방식에 비해 항목의 확장과 관리가 쉬우며 정확도와 표현력이 좋다.

패싯을 사용한 방법에서는 서로 다른 종류의 정보를 표현하는데 각각 한 개의 패싯을 사용한다. 여러 개의 패싯으로 구성된 패싯 목록((그림 1)에서 F_1, F_2, …, F_N)이 컴포넌트를 표현한다. 한 개의 패싯은 동일한 종류의 정보를 표현하는 다양한 패싯 값((그림 1)의 T_{11}, T_{1A}등)으로 구성된다. 목록에 있는 한 개의 패싯이 컴포넌트의 한 개의 특징을 표현한다. 컴포넌트는 한 개의 패싯에 대해 여러 개의 패싯 값을 가질 수 있다. (그림 1)이 컴포넌트 σ와 그것을 패싯 값으로 표현한 것을 보여 주고 있다.



(그림 1) 패싯에 의한 컴포넌트 표현

3. 소프트웨어 컴포넌트 서술자의 확장

2.1절에서 살펴 본 문제점에 대한 대안으로 본 논문에서 제안하는 것은 컴포넌트를 설명하는 표준적인 서술자를 컴포넌트와 함께 패키징하여 유통하고 배치하는 것이다.

컴포넌트를 유통하거나 컴포넌트를 배치할 때 컴포넌트에 관한 정보가 필요하다. 컴포넌트 제작자는 컴포넌트를 판매하기 위해 표준화된 형태로 컴포넌트에 관한 정보를 제공하고 또한 컴포넌트 구매자도 표준화된 컴포넌트의 정보를 얻을 수 있다면 컴포넌트의 판매 및 네트워크 상의 호스트들에 컴포넌트를 설치하는 배치가 쉬어지며, 판매자와 구매자의 생산성이 향상될 것이다. 또한 구입한 컴포넌트를 컴포넌트 저장소에 저장할 때 표준화된 정보를 이용하여 그 컴포넌트를 저장한다면 컴포넌트 저장소 관리가 용이해지며 후에 컴포넌트 검색을 쉽게 할 수 있다.

3.1 소프트웨어 컴포넌트 서술자(Software Component Descriptor)

소프트웨어 패키지는 서술자와 소프트웨어를 설치하는데 필요한 파일의 집합으로 구성된다. 서술자는 패키지의 특성을 표현하고 패키지에 있는 여러 파일에 대한 정보를 가지고 있다.

개방형 소프트웨어 서술자(Open Software Description, OSD)[24]는 Marimba와 MicroSoft사가 W3C에 제안한 것으로 XML에 기초하여 소프트웨어 패키지와 패키지들간의 종속성을 서술하는 어휘(vocabulary)를 제공한다.

컴포넌트 패키지는 소프트웨어 패키지의 한 종류이다. CORBA 3의 소프트웨어 패키징은 OSD를 확장한 소프트웨어 컴포넌트 서술자(SCD)[8]를 사용하여 컴포넌트의 정보를 기술한다. OSD, SCD, 그리고 다음 절에 서술할 확장된 SCD의 XML 요소를 <표 2>에 정리하였다.

〈표 2〉 컴포넌트 서술자의 요소들

OSD의 요소	SCD의 요소	확장된 SCD의 요소
s o f t p k g , implementation dependancy, title, license, os, osversion, processor, abstract, codebase, impltype, language, vm, memsize, disksize	OSD의 요소(softpkg, implementation, dependancy, title, license, os, osversion, processor) author, code, compiler, company, descriptor, fileinarchive, idl, humanlanguage, link, localfile, pkgtype, programminglanguage, propertyfile, repository, resource, runtime, threadsafety, webpage	SCD의 요소 전부 인터페이스 (type-def, const-def, except-def, interface-def, module-def 등 <표 3> 참조) domain function object medium

3.2 소프트웨어 컴포넌트 서술자의 확장

기본적으로 소프트웨어 컴포넌트 서술자는 여러 정보를 포함하고 있지만 컴포넌트를 표현하거나 또는 컴포넌트를 검색하는데 필요한 충분한 정보를 제공하지 못하므로 이 절

에서 이에 대한 확장을 한다. 확장되는 내용은 다음과 같다.

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

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

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

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

컴포넌트의 기능을 효과적으로 표현하는 방법으로 [9, 10]에서 제안한 (function, object, medium)의 3개의 값을 사용한다. Function은 컴포넌트가 수행하는 기본적인 기능 또는 행위를 표현한다. Object는 컴포넌트가 조작하는 대상 객체를 나타낸다. Medium은 컴포넌트의 행위가 수행되는 장소를 의미하는 것으로 기능을 지원하는 자료구조를 의미한다. 몇 가지 예를 들면 다음과 같다.

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

<표 3>의 DTD 정의는 이 논문에서 SCD에 새로 추가한 요소인 idl-spec, domain, comp-function을 표시하고 있다. idl-spec은 CORBA의 인터페이스를 표현하는 IDL을 기본으로 하여 만든 것이며 그 중의 일부만 표시하였다. 이 정의는 CORBA 컴포넌트와 CORBA 객체 이외에 EJB와 JavaBeans를 표현할 수 있는 일반성을 지니고 있다.

idl-spec 정의에 CORBA에만 고유하고 Java의 인터페이스에 없는 내용(예를 들면 메소드의 방향, 매개변수의 방향 등)을 포함하고 있는데 선택적인 사항으로 정의되어 있으므로 Java의 인터페이스를 표현하는데 문제가 없다. Java에만 있고 CORBA에 없는 package, import, 인터페이스와 메소드의 참조 수정자(access modifier)들을 선택 사항으로 정의하였다.

<표 4>에 CORBA의 IDL 예가 있으며 <표 5>에서 이를 XML 어휘를 사용하여 표시하였다. 또한 <표 6>에 EJB의 인터페이스의 예가 있으며 <표 7>에서 이를 XML 어휘로 표시하였다.

〈표 3〉 확장된 소프트웨어 컴포넌트 서술자를 위한 DTD

```
<!-- DTD for Extended Software Component Descriptor -->
<!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 accessmodifier (public | package) #IMPLIED >
<!ELEMENT super-interface (#PCDATA) >
<!ELEMENT method-def (method-direction?, return-type,
          parameter-def*, exception-def*) >
<!ATTLIST method-def name CDATA #REQUIRED >
<!ELEMENT method-direction EMPTY >
<!ATTLIST method-direction (oneway) #REQUIRED >
<!ELEMENT return-type (#PCDATA) >
<!ELEMENT parameter-def (parameter-direction?, parameter-
          type) >
<!ATTLIST parameter-def name CDATA #REQUIRED >
<!ELEMENT parameter-direction EMPTY >
<!ATTLIST parameter-direction value (in | out | inout) >
<!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) >
<!ELEMENT attr-def (attr-read*, attr-type) >
<!ATTLIST attr-def name CDATA #REQUIRED >
<!ELEMENT attr-type (#PCDATA) >
<!ELEMENT attr-read EMPTY >
<!ATTLSIT 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) >
```

〈표 4〉 CORBA의 인터페이스

```
interface Array {
  readonly attribute short height ;
  readonly attribute short width ;
  long get(in short row, in short col) ;
} ;
```

〈표 5〉 표 4의 인터페이스를 확장된 SCD로 표현

```
<?xml version = "1.0" ?>
<!DOCTYPE Array SYSTEM "idl-spec.dtd">
<idl-spec>
<interface-def>
<interface name = "Array">
<attr-def name = "height">
  <attr-read value = "readonly"/>
  <attr-type> short </attr-type>
</attr-def >
<attr-def name = "width">
  <attr-read value = "readonly"/>
  <attr-type> short </attr-type>
</attr-def>
<method-def name = "get">
  <return-type> long </return-type>
  <parameter-def name = "row">
     <parameter-direction value = in/>
     <parameter-type> short </parameter-type>
  </parameter-def>
  <parameter-def name = "col">
     <parameter-direction value = in/>
     <parameter-type> short </parameter-type>
  </parameter-def>
</method-def>
</interface>
</interface-def>
</idl-spec>
```

〈표 6〉 Java의 인터페이스

```
package com.book.cart ;
import java.util.* ;
import javax.ejb.EJBObject ;
import java.rmi.RemoteException ;
public interface Cart extends EJBObject {
  public void addbook(String title) throws RemoteException ;
}
```

〈표 7〉 표 6의 Java 인터페이스를 확장된 SCD로 표현

```
<?xml version = "1.0" ?>
<!DOCTYPE Array SYSTEM "idl-spec.dtd">
<idl-spec>
<package-def name = "com.book.cart"/>
<import-def name = "java.util.*"/>
<import-def name = "javax.ejb.EJBObject"/>
<import-def name = "java.rmi.RemoteException"/>
<interface-def>
<interface name = "Cart" accessmodifier = "public">
<super-interface> EJBObjec </super-interface>
<method-def name = "addbook" accessmodifier = "public">
  <return-type> void </return-type>
  <parameter-def name = "title">
     <parameter-type> String </parameter-type>
  </parameter-def>
  <exception-def name = "RemoteException"/>
</method-def>
</interface>
</interface-def>
</idl-spec>
```

3.3 확장된 소프트웨어 컴포넌트 서술자의 유용성

기존의 컴포넌트 저장소의 연구는 이미 컴포넌트에 대한 자료가 있다고 가정한다. 그러나 2.1절에서 살펴보았듯이 실제로는 컴포넌트 저장소에서 사용할 컴포넌트에 대한 정보를 얻는 방법은 쉽지 않으며 컴포넌트 및 프로그램의 종류에 따라 다르다. 따라서 이 논문에서는 다른 논문들에서 다루지 않았던 컴포넌트를 표현하는 방법으로 XML로 표시된 확장된 SCD를 제안하였다.

확장된 SCD는 코바 컴포넌트 이외에 EJB와 같은 다른 컴포넌트와 소프트웨어 패키지에 사용할 수 있다. Java의 인터페이스도 이 논문에서 제안된 확장된 SCD를 사용하여 표현할 수 있음을 3.2절에서 보였다. 인터페이스를 사용하지 않는 소프트웨어 패키지의 경우는 외부에서 사용할 수 있는 메소드에 대하여 그 메소드의 이름, 반환형, 매개 변수 등을 확장된 SCD를 사용하여 표현하고 또한 패키지에 대한 다른 일반 정보도 확장된 SCD를 사용하여 표현이 가능하다.

일반적으로 컴포넌트는 ZIP의 형태로 압축하여 패키징된다. 확장된 SCD의 확장자는 scd로 하여 ZIP 파일의 meta-info 폴더에 저장하여 유통함으로 컴포넌트에 관한 정보를 쉽게 얻을 수 있다.

<표 8>에서 확장된 SCD와 기존의 방법을 요약하여 비교하였다.

〈표 8〉 기존의 방법과 확장된 SCD의 비교

항 목	기존의 방법	확장된 SCD
컴포넌트의 표현	비표준적인 방법	표준적인 방법
자료의 공유	어려움	XML로 표현하여 용이함
인터페이스의 접근	인터페이스가 별도로 존재하여 어려움	인터페이스가 포함되어 용이함
패키징	서술자가 컴포넌트와 독립적으로 존재	서술자가 컴포넌트와 함께 패키징
적용 대상	각 컴포넌트에 독립적인 적용	다양한 컴포넌트 종류에 적용 가능
컴포넌트 저장소 검색에 사용	어려움	용이함(3.4절 참조)
생산성	낮음	향상

3.4 추출한 패싯

소프트웨어 컴포넌트 서술자는 나열 방법[18] 등에서 요구하는 별도의 분류나 조작없이 용이하게 패싯으로 전환될 수 있다. 즉 ELEMENT에 표현된 내용과 ATTRIBUTE에 있는 내용을 한 개의 패싯으로 만들 수 있다. 예를 들면 XML로 표시된 <function> delete </function>에서 function이 한 개의 패싯이 되고 delete가 패싯 값이 된다.

패싯 목록은 컴포넌트가 하는 일과 그 일을 하는 방법을 집약하여 표현할 수 있어야 하며 컴포넌트의 분류와 검색에 필요한 충분한 정보를 가지고 있으면서 또한 단순 명료하여야 한다[9]. 이런 원칙에 의거하여 3.1절과 3.2절에 서술한 내용에서 추출한 패싯을 아래에 서술하였다.

(1) 이름(name) : 컴포넌트의 이름을 표시한다.

(2) 버전(version) : 컴포넌트의 버전을 표시한다. 버전을 표시하는 형태는 중요한 버전과 부수적인 버전들을 숫자와 쉼표로 표시한다. 예를 들면 1,0,0,0과 같다.
(3) 패키지 타입(pkgtype) : 컴포넌트의 종류를 나타낸다. SCD에 "CORBA Component"와 "CORBA Interface Impl"이 CORBA 컴포넌트와 CORBA 객체를 위해 예약어로 지정되어 있다. 자바를 위해서 "JavaBeans"와 "EJB"를, 일반 프로그램을 위해 "others"를 추가한다.
(4) 인터페이스 : 3.2절에서 정의한 메소드 이름과 예외 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 패싯이다.
(8) 컴파일러(compiler) : 코드를 생성한 컴파일러를 지정한다.
(9) 종속성(dependency) : 수행할 때 환경적인 종속성을 표현한다. 예를 들면 ORB를 지정할 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 지정한다. 예로 Java VM과 같은 것이다.

4. 컴포넌트 저장소의 검색

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

4.1 신경 접속 행렬

신경망 기술은 지식의 표현, 추론과 규칙 확장에 많이 사용되고 있다[25]. 이 논문에서 사용하는 신경망[11]은 기존에 제안된 신경 연상 기억장치를 컴포넌트 검색에 적용하였다. 컴포넌트를 검색하는데 $N \times F$ 의 신경 접속 행렬(synaptic connectivity matrix)[11], W를 사용하는데 행렬의 열은 컴포넌트를, 행은 패싯을 표현한다.

이 장에서 사용하는 변수들은 <표 9>에 정의되어 있다.

〈표 9〉 변수들

변 수	정 의
N	저장소에 있는 컴포넌트의 개수
n_i	패싯 값 i를 가지고 있는 컴포넌트의 수
F	다른 패싯 값의 개수

정의 1 : 신경 접속 행렬 W의 원소 w_{ij}

$$w_{ij} = 1 \ \textit{if componenet } j \textit{ has facet value } i$$
$$= 0 \ \textit{otherwise}$$

정의 2 : 사용자 질의어 벡터 Q의 원소 $q_i (1 \le i \le F)$

$$q_i = 1 \ \textit{if a component with facet value } i \textit{ is needed}$$
$$= 0 \ \textit{otherwise}$$

정의 3 : 검색 출력 벡터 O

$$O = Q \cdot W$$

$Q \cdot W$를 계산하면 N개의 원소를 가진 출력 벡터 O가 만들어지는데 o_i 값이 컴포넌트 i의 검색 만족도가 된다.

예 1. 신경 접속 행렬

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

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

사용자가 "delete"라는 패싯 값을 갖는 컴포넌트를 찾는다고 하자. 그러면 질의어 Q와 이에 대한 출력 벡터는 다음과 같다.

$$Q = [0,1,0], \quad O = Q \cdot W = [0,1,1,0]$$

결과는 컴포넌트 2와 3이 선택된다.

여러 컴포넌트가 동일한 패싯 값을 가질 수 있으며 또한 한 컴포넌트가 여러 개의 다른 패싯 값을 가질 수 있다. 이 때 컴포넌트가 가지는 패싯 값의 희소성과 컴포넌트가 검색하는 패싯 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.

$$w'_{ij} = w_{ij} \cdot \frac{\log_2(N/n_i)}{\sum_{z=1}^{F} w_{zj} \log_2(N/n_z)} \quad s.t. \quad \sum_{i=1}^{F} w'_{ij} = 1$$

예 2. 가중치 신경 접속 행렬

예 1의 행렬 W에 가중치 함수를 적용하여 아래와 같은 가중치 신경 접속 행렬 W'를 구한다.

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

4.2 논리 연산자를 이용한 검색

앞 절의 가중치 함수는 한 개의 패싯 값에 대해서는 작동하지만 두 개 이상이 되면 문제가 될 수 있다. 이에 대한 예를 살펴보면 다음과 같다.

예 3. 논리 관계를 고려하지 않은 검색

질의어 Q = [0, 1, 1]에 대한 출력 벡터는 다음과 같다.

$$O = Q \cdot W = [0, 2, 1, 1],\ O' = Q \cdot W' = [0, 2/3, 1, 1]$$

컴포넌트 2의 경우는 2개의 패싯 값을 만족하지만 패싯 값 1개를 만족한 컴포넌트 3과 4보다도 작은 출력값을 가지게 된다. 특히 패싯 값 2개를 동시에 만족(AND)하는 경우의 질의였다면 오히려 컴포넌트 2만 질의를 만족하고 컴포넌트 3과 4는 질의를 만족할 수 없다.

예 3의 문제는 [11]에서 제안한 검색 방법이 패싯 값간에 논리 관계를 고려하지 않고 논리합 관계만 다루고 있는데 문제가 있다. 패싯 검색을 사용한 다른 논문들[9-12]에서도 논리 연산자를 사용한 검색을 고려하지 않고 있다. 논리 연산자를 사용한 검색은 기본적으로 필요하며 CORBA의 트레이딩 객체 서비스[26]에서도 요구하고 있다. 이 절에서 패싯 값간의 논리 관계를 고려한 검색을 제안한다.

4.2.1 논리합 검색

논리합 검색의 출력 벡터는 다음과 같이 정의한다.

정의 5 : 논리합 검색의 출력 벡터 O_{OR}

$$O_{OR} = Q \cdot W'$$

예 4. 논리합 검색의 예

논리합 검색을 위한 질의어가 Q = [0, 1, 1]이면 출력 벡터는 다음과 같이 계산된다.

$$O_{OR} = Q \cdot W' = [0, 2/3, 1, 1]$$

이 때 컴포넌트 2가 컴포넌트 3과 4보다 만족도가 떨어지는데 이는 컴포넌트 2가 사용자가 원하지 않는 다른 기능을 제공하기 때문이다.

4.2.2 논리곱 검색

논리곱 검색을 위해 검색하는 모든 패싯 값에 대하여 컴포넌트가 가지고 있는 패싯 값의 개수를 가진 히트율을 계산한다.

정의 6 : 질의어에 대한 히트율 벡터 H

어떤 질의어에 대한 히트율 벡터 H의 원소 h_j $(1 \le j \le N)$는 다음과 같이 정의한다.

$$h_j = \frac{f_j}{f},\ \text{where}\ f = \sum_{i=1}^{F} q_i,\ f_j = \sum_{i=1}^{F} q_i \cdot w_{ij}$$

정의 7 : 논리곱 검색의 출력 벡터 O_{AND}의 i번째 원소 $o_{AND\,i}$

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

즉 O_{OR}의 i번째 원소와 히트율 벡터의 i번째 원소의 곱으로 구한다.

예 5. 논리곱 검색

질의어 Q = [0, 1, 1]에 대하여 히트율 벡터와 최종 검색 결과는 아래와 같다.

$$H = [0, 1, 1/2, 1/2]$$
$$O_{AND} = [0 \cdot 0, 1 \cdot 2/3, 1/2 \cdot 1, 1/2 \cdot 1]$$
$$= [0, 2/3, 1/2, 1/2]$$

결과에서 보듯이 컴포넌트 2가 2개의 패싯 값을 동시에 만족하므로 한 개의 패싯 값만 만족하는 컴포넌트 3과 4보다 만족도가 높게 나온다.

4.2.3 논리부정 검색

신경 접속 행렬에서 k행에 표시된 패싯 값을 포함하지 않는 컴포넌트를 검색하는 것은 기본 신경 접속 행렬에서 $w_{kj}=0$이면 컴포넌트 j가 선택되고 $w_{kj}=1$이면 컴포넌트 j가 제외되는 것을 의미한다. 논리부정 검색에 대해서도 일반 검색과 동일하게 벡터와 행렬의 곱으로 만족하는 컴포넌트를 찾아내기 위해 새로운 행렬을 다음과 같이 정의한다.

정의 8 : 논리 부정 검색(k번째 패싯 값을 포함하지 않는 검색)을 위한 신경 접속 행렬 $\overline{k}W$

W에 대해, k번째 패싯 값을 포함하지 않는 컴포넌트를 검색할 때 사용되는 신경 접속 행렬을 $\overline{k}W$로 표시하자. $\overline{k}W$의 원소인 $\overline{k}w_{ij}$는 다음과 같이 정의한다.

$$\overline{k}w_{ij} = \begin{cases} 1 - w_{ij}, & \text{if } i = k \\ w_{ij}, & \text{otherwise} \end{cases}$$

정의 9 : 논리부정 검색을 위한 가중치 신경 접속 행렬 $\overline{k}W'$

k번째 패싯 값을 포함하지 않는 컴포넌트를 검색할 때 사용되는 가중치 신경 접속 행렬의 원소 $\overline{k}w'_{ij}$는 다음과 같이 정의한다.

$$\overline{k}w'_{ij} = \overline{k}w_{ij} \cdot \frac{\log_2(N/n_i)}{\sum_{z=1}^{F} \overline{k}w_{zj} \log_2(N/n_z)}$$

예 6. 논리 부정 검색

질의어 Q = [0, $\overline{1}$, 0]가 있다. $\overline{1}$은 논리 부정 검색을 의미한다. k가 2이므로 W에서 $\overline{2}W$와 $\overline{2}W'$를 다음과 같이 얻는다.

$$\overline{2}W' = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \overline{2}W' = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \end{bmatrix}$$

부정 검색 결과는 다음과 같다.

$$O_{NOT} = Q \cdot \overline{2}W' = [1/2, 0, 0, 1/2]$$

논리부정 검색을 하기 위해 가중치 신경 접속 행렬 $\overline{k}W'$ 전체를 새로 만드는 것은 복잡도가 $O(N \cdot F)$이다. 또한 한 개의 원소인 $\overline{k}w'_{ij}$를 구하는 복잡도는 $O(F)$이다. 복잡도를 향상시키기 위해 다음과 같은 벡터 S를 정의한다.

정의 10 : 가중치 합 벡터 S

S의 원소 s_j는 다음과 같이 정의한다.

$$s_j = \sum_{i=1}^{F} w_{ij} \cdot \log_2(N/n_i) \quad (1 \le j \le N)$$

이와 같이 정의된 S를 미리 계산하여 저장하고 있으면 패싯 값 k를 부정한 신경 접속 행렬의 원소 $\overline{k}w'_{ij}$는 다음과 같이 구할 수 있다.

신경 접속 행렬의 원소 $\overline{k}w'_{ij}$를 구하는 알고리즘

```
if w'_kj = 0
  if i ≠ k
    k̄w'_ij = w'_ij · s_j / (s_j + log2(N/(N − n_k)))
  else
    k̄w'_ij = log2(N/(N − n_k)) / (s_j + log2(N/(N − n_k)))
else
  if i ≠ k
    k̄w'_ij = w'_ij · s_j / (s_j − log2(N/n_k))
  else
    k̄w'_ij = 0
```

위의 알고리즘을 사용하면 $\overline{k}w'_{ij}$를 구하는 복잡도가 $O(F)$에서 $O(1)$으로 향상된다.

예 7. 논리부정 검색을 위한 신경 접속 행렬

예 1의 W에 대한 가중치 합 벡터 S는 다음과 같다.

$$S = [1, 3, 1, 1]$$

질의어 $Q = [0, \overline{1}, 0]$,에 대한 $\overline{2}W'$의 원소를 위의 알고리즘을 사용하여 아래와 같이 구할 수 있으며 예 6의 결과와 동일함을 알 수 있다.

$$\overline{2}w'_{11} = \frac{1 \cdot 1}{1 + \log_2 2} = \frac{1}{2}, \quad \overline{2}w'_{12} = \frac{(1/3) \cdot 3}{3 - \log_2 2} = \frac{1}{2}$$

$$\overline{2}w'_{13} = 0, \quad \overline{2}w'_{14} = 0$$

$$\overline{2}w'_{21} = \frac{\log_2 2}{1 + \log_2 2} = \frac{1}{2}, \quad \overline{2}w'_{22} = 0, \quad \overline{2}w'_{23} = 0$$

$$\overline{2}w'_{24} = \frac{\log_2 2}{1 + \log_2 2} = \frac{1}{2}$$

$$\overline{2}w'_{31} = 0, \quad \overline{2}w'_{32} = \frac{(1/3) \cdot 3}{3 - \log_2 2} = \frac{1}{2}, \quad \overline{2}w'_{33} = 0$$

$$\overline{2}w'_{34} = \frac{1 \cdot 1}{1 + \log_2 2} = \frac{1}{2}$$

이제 논리 연산자가 동시에 사용될 때의 예를 살펴보자.

예 8. NOT과 OR의 예

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

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

예 9. NOT과 AND의 예

$Q_{AND} = [1, \overline{1}, 0]$이면 히트율 벡터와 최종 검색 결과는 다음과 같다.

$$H = [1, 1/2, 0, 1/2]$$
$$O_{AND} = [1 \cdot 1, 1/2 \cdot 1/2, 0 \cdot 0, 1/2 \cdot 1/4, 0, 1/4]$$

4.3 신경 접속 행렬의 계산의 복잡도

패싯 값 전체의 개수는 상당히 많으나 일반적으로 한 개의 검색에 사용하는 패싯 값은 매우 적다. 따라서 행렬 전체를 연산할 필요없이 행렬에서 검색하는 패싯 값의 행 벡터만 선택하여 연산한다. 즉 질의어 Q에서 0에 해당하는 행렬의 행은 무시하고 1이나 $\overline{1}$로 표시된 행만 선택한다. $\overline{1}$로 표시된 패싯 값에 대해서는 $\overline{k}w'_{ij}$를 구하는 알고리즘에서 정의한 방법을 사용하여 선택된 행에 대해서만 $\overline{k}w'_{ij}$를 구하면 되므로 논리부정 검색을 위한 가중치 신경 접속 행렬을 구하는 복잡도는 $O(N)$이다. 따라서 행렬 연산을 포함한 전체적인 복잡도는 $O(N)$으로 기존의 연산 복잡도, $O(N \cdot F)$[11]에서 향상되었다.

4.4 동의어 사전

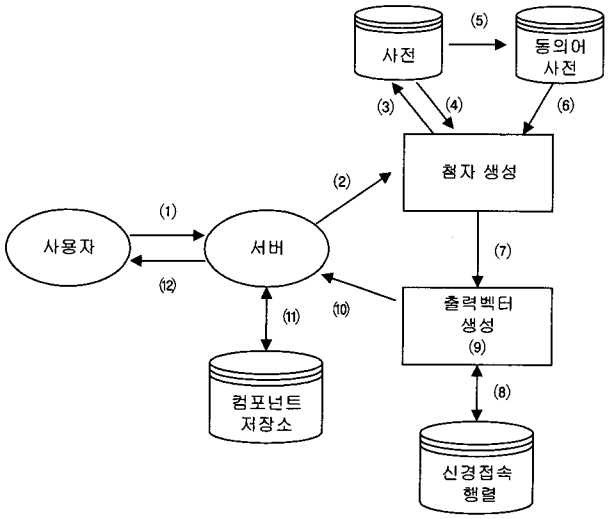
한 사물을 표현하는데 동의어를 사용하여 다르게 표현할 수 있다. 따라서 어떤 패싯 값으로 정의된 컴포넌트가 없더라도 그 패싯 값의 동의어로 정의된 컴포넌트를 찾을 수 있다. 이를 지원하기 위한 것이 동의어 사전이다. 동의어를 지원하는 여러 시스템들이 있는데 이 논문에서는 2중 구조의 동의어 사전을 제안하여 효율적인 검색을 지원한다.

(1) 패싯 값 사전 : 이 사전은 현재 컴포넌트 저장소에 저장되어 있는 컴포넌트가 소유하고 있는 패싯 값들을 가지고 있다. 또한 각 패싯 값에 대한 인덱스를 가지고 있다.

(2) 동의어 사전 : 동의어 사전은 패싯 값에 대한 동의어를 포함하고 있다. 동의어 사전에 있는 각 동의어 항목은 패싯 값 사전에 있는 자신의 동의어인 패싯 값을 가리키고 있으며 그 패싯 값과의 적합도(relevance)를 표시하는 값 $r(0 \le r \le 1)$을 가지고 있다.

4.5 컴포넌트 검색 과정

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



(그림 2) 컴포넌트 검색 시스템 구조

(1) 사용자가 원하는 패싯 값을 입력하여 해당되는 컴포넌트의 검색을 요청한다.

(2) 검색 서버는 고객의 요청을 받아 해석하고 그 내용을 첨자 생성자에게 보낸다.

(3) 첨자 생성자는 고객이 입력한 패싯 값을 패싯 값 사전에 보낸다.

(4) 사전에 패싯 값이 존재하면 첨자(i)를 구하여 첨자 생성자에게 보낸다.

(5) 만약 패싯 값 사전에 패싯 값이 없으면 동의어 사전에서 해당하는 패싯 값이 있는지 확인한다.

(6) 만약 있으면 동의어 항목이 가리키는 패싯 값 사전에 있는 패싯 값의 첨자(k)를 구하여 첨자와 적합도(r)을 보낸다.

(7) 사전에서 얻은 첨자와 적합도를 출력벡터 생성자에게 보낸다.

(8) 넘겨받은 각 첨자 i에 대해 신경 접속 행렬의 i번째 행 Wi를 추출한다. 동의어 사전에서 온 첨자 k에 대해서는 신경 접속 행렬의 k번째 행을 추출하여 $Wk = r \cdot Wk$를 구한다.

(9) 컴포넌트 생성자가 추출된 모든 행을 사용하여 컴포넌트의 출력 벡터를 구한다. 출력 벡터의 연산은 4.2에서 보인 것처럼 논리 연산자에 따라 계산 결과가 달라진다.

(10) 출력 벡터를 서버에 보낸다.

(11) 출력 벡터를 사용하여 컴포넌트 저장소에서 컴포넌트에 대한 정보를 얻는다.

(12) 출력 벡터의 값이 큰 순서대로 컴포넌트를 정렬하여 컴포넌트의 정보를 사용자에게 표시한다.

5. 결론 및 향후 연구 방향

본 논문에서 인터페이스와 컴포넌트의 기능 등을 XML 어휘로 표현하였고 이를 CORBA 3의 소프트웨어 컴포넌트 서술자에 포함하여 서술자를 확장하였다. 이 확장된 서술자는 CORBA 컴포넌트, CORBA 객체, EJB, JavaBeans 및 일반 프로그램 등에 적용할 수 있다. 확장된 소프트웨어 컴포넌트 서술자에서 유용한 정보를 추출하여 이를 패싯으로 만들고 컴포넌트 저장소에 있는 컴포넌트를 검색하는데 사용하였다. 또한 확장된 소프트웨어 컴포넌트 서술자를 컴포넌트와 함께 패키징하여 유통 및 배치할 것을 제안하였다.

이렇게 함으로 복잡한 프로그래밍 과정이나 인터페이스 저장소에 의존하던 기존의 방법을 탈피하여 어떤 종류의 컴포넌트에 대해서도 표준적이고 쉬운 방법으로 인터페이스에 관한 정보를 얻을 수 있다. 또한 컴포넌트에 관한 일반적인 정보나 컴포넌트 저장소의 검색에 필요한 정보도 확장된 소프트웨어 컴포넌트 서술자를 통하여 용이하게 얻을 수 있다. 컴포넌트 개발자 및 사용자들이 확장된 소프트웨어 컴포넌트 서술자를 표준화하여 사용하면 컴포넌트 검색뿐 아니라 컴포넌트의 유통 및 배치에도 많은 혜택이 될 것이다.

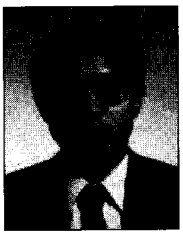
본 논문에서 제안한 패싯 검색 방법은 기존의 패싯 방법에서 지원되지 않은 논리 연산자를 사용한 검색이 가능하게 하였다. 또한 논리 부정 검색을 위한 가중치 신경 접속 행렬을 계산하는 알고리즘을 개발하고 검색하는 패싯 값의 첨자를 사용하여 검색 시간의 복잡도를 향상하였다.

향후 연구 과제로 이 논문에서 제안한 시스템을 구현하고 기존의 다른 시스템과 비교·평가한 실험결과를 추가하는 일이다. 또한 컴포넌트 저장소의 검색에서 이 논문에서 제안한 논리 연산자를 사용한 검색뿐 아니라 CORBA의 트레이딩 객체 서비스에서 요구하는 다양한 검색이 지원될 수 있는 방법이 요구된다.

참 고 문 헌

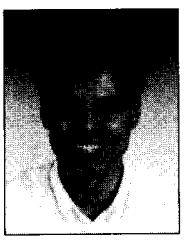
[1] P. Freeman, "Reusable Software Engineering : Concepts and Research Directions," Proc. Workshop Reusability in

Programming, pp.2-16, 1983.
[2] W. P. Frakes and T. P. Pole, "An Empirical Study of Representation Methods for Reusable Software Components," IEEE Transaction on Software Engineering Methodology, Vol.20, No.8, pp.617-630, Aug., 1994.
[3] Christopher G. Drummond, Dan lonescu, and Robert C. Holte, "A Learning Agent that Assists the Browsing of Software Libraries," IEEE Transactions on Software Engineering, Vol.26, No.12, pp.1179-1196, 2000.
[4] J. Guo and Luqi, "A survey of Software Reuse Repositories," Proceedings of the 7th IEEE Int. Conf. & W/S on the Engineering of Computer Based System, pp.88-96, 2000.
[5] Rober C. Seacord et al, "AGORA : A Search Engine for Software Components," IEEE Internet Computing, pp.62-70 Nov · Dec., 1998.
[6] 금영욱, "CORBA 컴포넌트의 구현", 정보처리학회지, 제7권 제4호, pp.60-69, 2000.
[7] 금영욱, 장연세, "코바 3 프로그래밍 바이블", 도서출판 그린, 2000.
[8] CORBA Component Model : http : //www.omg.org/cgi-bin/doc?ptc/99-10-03.pdf.
[9] R. Prieto-Diaz, "A Software Classification Scheme," Doctorial Dissertation, Department of Computer Science, University of California, Irvine, California, 1985.
[10] R. Priesto-Diaz, "Implementing Faceted Classification for Software Reuse," CACM, Vol.34, No.5, pp.89-97, May, 1991.
[11] Zhiyuan Wang, "Component-Based Software Engineering," Doctorial Dissertation, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, New Jersey, 2000.
[12] Hsian-Chou Liao et al, "Using a Hierarchical Thesaurus for Classifying and Searching Software Libraries," Proceedings of the COMPSAC '97, pp.210-216, 1997.
[13] Enterprise JavaBeans(TM) 2.0 Specification : ftp : //ftp.java.sun.com/pub/ejb/947q9tbb/ejb-2_0-fr2-spec.pdf.
[14] JavaBeans 1.01 specification : ftp : //ftp.javasoft.com/docs/beans/beans.101.pdf, ftp : //ftp.java.sun.com/pub/j2ee/43098h44w/j2ee-1_3-pfd4-spec.pdf.
[15] 금영욱, "CORBA - 분산 객체 통합 기술", 한국정보과학회 소프트웨어공학회지, 제12권 제2호, pp.5-14, 1999.
[16] P. Devanbu, R. Brachman, P. Selfridge, and B. Ballard, "LaSSIE : A Knowledge-Based Software Information System," Comm. ACM, Vol.34, No.5, pp.34-49, May, 1991.
[17] E. Ostertag, J. Hendler, R. Priesto-Diaz, and C. Braun, "Computing Similarity in a Reuse Library System : An AI-Based Approach," ACM Transaction on Software Engineering Methodology, Vol.1, No.3, pp.205-228, July, 1992.
[18] M. Dewey, "Decimal Classification and Relative Index," 19th ed., Forest Press Inc., Albany, New York, 1979.
[19] W. P. Frakes and B. A. Nejmeh, "An Information System for Software Reuse," in W. Tracz (editor), Software Reuse : Emerging Technology, IEEE Computer Society Press, Monterey, California, 1988.
[20] J. Jeng and B. Cheng, "A formal Approach to Reusing More General Components," Proc. of IEEE 9th Knowledge-Based Software Engineering Conference, pp.90-97, Monterey, California, Sep., 1994.
[21] R. Mili, A. Mili and R. Mittermeir, "Storing and Retrieving Software Components : A Refinement Based System," IEEE Transactions on Software Engineering, Vol.23, No.7, pp.445-460, 1997.
[22] F. R. Campagnoni and K. Ehrlich, "Information Retrieval Using a Hypertext-Based Help System," Proc. 12th Int'l Conf. Research and Development in Information Retrieval, pp.212-220, 1989.
[23] R. H. Thomson and W. B. Croft, "Support for Browsing in an Intelligent Text Retrieval System," Int'l J. Man-Machine studies, Vol.30, pp.639-668, 1989.
[24] The Open Software Description Format (OSD), http : //www.w3.org/TR/NOTE-OSD.html.
[25] F. J. Kurfess, "Neural Networks and Structured Knowledge," Special Issue of Journal of Applied Intelligence, Vol.11, No.1, pp.135-146, 1999.
[26] CORBA Trading Object Service, ftp : //ftp.omg.org/pub/docs/formal/00-06-27.pdf.



금 영 욱

email : ywkeum@sungkyul.edu
1978년 서울대학교 수학과(이학사)
1992년 서강대학교 정보처리학과(이학석사)
1997년 서강대학교 전자계산학과(공학박사)
1978년~1981년 (주)대우 시스템즈 엔지니어
1982년~1993년 한국 및 미국 IBM 시스템즈 엔지니어
1997년~현재 성결대학교 컴퓨터학부 조교수
관심분야 : 분산 · 병렬처리, 분산객체기술, 컴포넌트 기술



박 병 섭

e-mail : bspark@woosuk.ac.kr
1989년 충북대학교 컴퓨터공학과(공학사)
1991년 서강대학교 전자계산학과(공학석사)
1997년 서강대학교 전자계산학과(공학박사)
1997년~1999년 국방과학연구소 선임연구원
2000년~현재 우석대학교 컴퓨터교육과 교수
관심분야 : ATM switch design, Wireless Network, Mobile-IP, 컴포넌트 기술