Case 기반 재사용에서 효율적인 의미망의 구축과 컴포넌트 검색

Construction of Efficient Semantic Net and Component Retrieval in Case-Based Reuse

한정수
백석대학교 정보통신학부

Jung-Soo Han(jshan@bu.ac.kr)

요약

본 연구는 객체 지향 소스 코드의 검색과 재사용을 효율적으로 수행할 수 있는 의미망을 구축하였다. 이를 위하여 각 노드 간 객체지향 상속의 개념을 표현할 수 있도록 의미망의 초기 관련값을 시소러스로 구축하였다. 또한, 의미망의 노드와 간선을 활성화시키고 활성값을 전파시키기 위해 사용되는 스프레딩 엑티베이션 방법의 단점을 보완하여 스프레딩 엑티베이션의 성능은 최대한 유지하면서 검색 속도를 향상시킬 수 있는 방법을 제안하였다.

■ 중심어 : | 케이스 | 의미망 | 시소러스 | 스프레딩 엑티베이션 |

Abstract

In this paper we constructed semantic net that can efficiently conform retrieval and reuse of object-oriented source code. In order that initial relevance of semantic net was constructed using thesaurus to represent concept of object-oriented inheritance between each node. Also we made up for the weak points in spreading activation method that use to activate node and line of semantic net and to impulse activation value. Therefore we proposed the method to enhance retrieval time and to keep the quality of spreading activation

■ keyword : | Case | Semantic Net | Thesaurus | Spreading Activation |

I. 서 론

최근 20년 간 소프트웨어 재사용에 대해 많은 노력들이 이루어져 왔으며, 특히 Case-Based Reasoning (CBR) 시스템에 대한 다양한 연구가 이루어져 왔다. CBR 사이클의 검색(retrieval) - 재사용(reuse) - 수정(revise) - 유지(retain) 중 재사용을 위해 가장 핵심이 되는 부분이 컴포넌트 검색인데, 이는 검색 대상에 따라 컴포넌트 기술 방법과 구축 방법 그리고 검색 방법이 크게 달라진다. 또한 컴포넌트를 검색하는 방법에 따라 사용자의 요구사항을 얼마나 반영할 수 있는가와 라이브러리 확장성 등이 결정되기 때문에 검색 방법은 매우 중요하다. 특히 소스 코드를 재사용하기 위한 Case 기반 검색에 있어서 의미망의 효율적인 구성은 무엇보다 중요하다[1].

이에 본 연구에서는 의미망의 노드와 간선을 활성화시키고 활성값을 전파시키기 위해 사용되는 스프레딩 엑티베이션 방법의 단점을 보완하여 스프레딩 엑티베이션의

접수번호 : #060118-003
접수일자 : 2006년 01월 18일
심사완료일 : 2006년 02월 14일
교신저자 : 한정수, e-mail : jshan@bu.ac.kr

성능은 최대한 유지하면서 검색 속도를 향상시킬 수 있는 검색 시스템을 설계하고자 한다[2]. 또한, 각 노드 간 객체지향 상속의 개념을 표현할 수 있도록 초기 관련값을 시소러스로 구축하고자 한다. 이때, 각 Case를 구성하는 클래스들을 상속관계에 따라 개념적으로 분류하였고, 시소러스 방법에 퍼지 논리를 적용하여 객체지향 시소러스를 구축한다.

본 연구의 의미망 구축은 크게 2가지 관점에서 재사용에 유용하다. 하나는 프로그래머의 관심을 라이브러리 내에 있는 컴포넌트로 유도하여 재사용성을 높일 수 있다. 이는 검색된 컴포넌트 내에 각 클래스를 하이퍼링크로 라이브러리 API와 연결시키는 방법 등을 사용하여 재사용을 유도할 수 있다. 다른 하나는, 여러 다양한 클래스들이 포함된 전형적인 유형의 프로그래밍 패턴을 제공함으로써 프로그래머로 하여금 프로그램의 가이드라인으로 사용할 수 있도록 도움을 준다.

본 연구는 서론에 이어 제2장에서는 case 기반 재사용에 대하여 고찰하며, 제3장에서는 의미망 구축에 대하여 기술하였고, 제4장에서는 개선된 검색방법을 기술하고, 제6장에서 실험결과를 평가하였으며, 끝으로 결론을 맺는다.

II. case 기반 재사용

많은 CBR 시스템에서 라이브러리 내에 있는 각 클래스를 Case로 취급하고 있으며, 또한 이러한 Case를 재사용 단위인 컴포넌트화 하고 있다[3]. Case는 각 컴포넌트의 소스 코드를 가지고 있으며, 그 컴포넌트를 표현할 수 있는 속성이나 특징을 포함할 수 있다. 또한 컴포넌트의 행위적 특성을 텍스트로 포함할 수 있다[4]. [그림 1]은 자바로 구현된 컴포넌트의 Case 예이다. 'URL_Reader' 컴포넌트는 BufferedReader를 사용해서 URL을 직접 읽어오는 방법에 대한 소스이다. 만약, 어떤 사용자(프로그래머)가 BufferedReader를 사용해서 URL을 읽어오는 프로그램을 작성고자 할 때 프로그램 작성 방법을 모르거나 작성 패턴을 알고 싶을 때, 사용자는 'BufferedReader' 와 'URL'과 같이 라이브러리에 있는 클래스 이름을 이용한 소스 코드를 그대로 질의함으로써 'URL_Reader' 컴포넌트를 검색할 수 있다. Case 기반 검색은 크게 2 단계로 이루어진다. 1 단계는 의미망(semantic net)을 구성하는 단계이다. 라이브러리에 저장된 클래스를 기반으로 한 컴포넌트는 파싱 과정을 거쳐 의미망을 구성한다. 의미망은 소스 코드의 'class' 등으로 구성된 노드와 'relevance' 등으로 구성된 간선으로 만들어 진다. 2 단계는 의미망을 이용한 검색이다. 사용자가 질의로 준 소스 코드를 이용한 검색 단계이다. 질의 소스 코드에서 추출된 식별자가 의미망을 활성화시켜 연관된 컴포넌트를 검색한다.

Source Code

```
import java.net.*;
import java.io.*;

public class URL_Reader
{
  public static void main(String[] args)
  {
    URL kbs=new URL("http://www.kbs.co.kr");
    BufferedReader in=new BufferedReader(
      new InputStreamReader(kbs.openStream()));
    String inputLine;
    while ((inputLine=in.readLine())!=null)
      Sysem.out.println(inputLine);
    in.close();
  }
}
```

Goal Text

- *How to read directly from a URL using BufferedReader?*
- *How to copy directly from a URL to an output stream?*

그림 1. Case-URL_Reader

III. 의미망의 구축

1. 의미망

Case 단위로 라이브러리에 저장된 각 컴포넌트는 파싱 단계를 거쳐 파싱 트리를 형성한다. 'class',

'interface', 'method', 그리고 'variable' 등을 식별자로 추출하였다. 추출 과정은 컴포넌트 소스 코드를 읽어 각 식별자를 추출하고 식별자 사이의 관계 정보를 저장한다. 의미망은 파싱 트리로부터 만들어진다. 파싱으로부터 생성된 'class', 'interface', 'method', 'variable'은 의미망에서 노드로 구성되며, 'relevance', 'subclass', 'implements', 'member', 그리고 'invoke' 등은 클래스들 간의 관계로 취급하여 노드 간 간선으로 구성된다. [그림 1]의 Case-URL_Reader에 대한 의미망은 [그림 2]와 같이 구성된다.

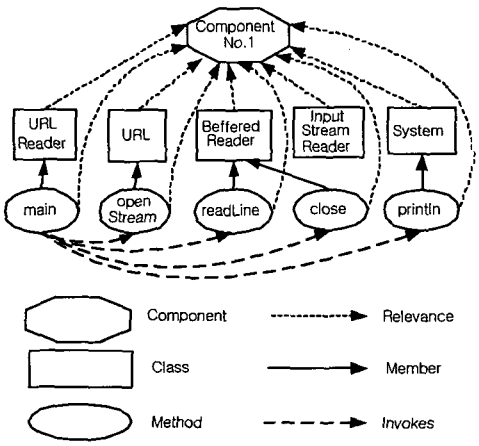

그림 2. 'URL_Reader' 컴포넌트의 의미망

컴포넌트 검색은 사용자에 의해 주어진 질의가 의미망을 활성화시킴으로써 이루어진다. 사용자는 자신이 작성하고 있는 소스 코드를 질의로 사용한다. 질의로 주어진 소스 코드는 구문 분석을 통해 식별자로 추출되는데, 사용자 소스 파일을 입력받아 먼저 토큰 단위로 식별자 이름을 모두 추출한 후, 처음 식별자명에서 다음 식별자명이 나타날 때까지 비교하면서 식별자 수만큼의 반복으로 정보를 추출한다. 추출된 식별자는 의미망에 있는 노드와 비교하기 위하여 inexact string matching algorithm 등을 이용한다[1].

소스 코드의 식별자는 각 컴포넌트의 의미망에 있는 노드와 비교되어 매치되는 노드를 활성화시킨다. 노드의 활성화는 스프레딩 엑티베이션 알고리즘에 의해 이루어진다[2]. 의미망의 노드와 간선의 초기 활성값을 이용하여 서로 연결되어 있는 노드를 참조해 가면서 활성값을 계산하게 된다. 순환이 반복될수록 활성값은 안정되며 활성값이 기준에 미달되는 부분은 자동으로 제거되어 계산과정이 종료된다. 최종적으로 활성값이 가장 높은 컴포넌트들이 검색된다.

2. 의미망 구축을 위한 요구사항

Case 기반 컴포넌트 검색을 위해 효율적인 의미망을 구축하기 위해서는 객체지향 패러다임을 검색에 적합한 형태로 해석·적용하고, 기존의 시소러스에서 이용된 관계성을 클래스와 컴포넌트의 관계로 재정의함으로써 의미망의 구축과 검색을 위한 새로운 기본 구조가 만들어져야 한다. 또한 구축된 의미망에 효율적인 검색 메커니즘을 적용함으로서 원하는 컴포넌트를 쉽게 검색할 수 있다. 기존 시소러스의 문제점들과 스프레딩 엑티베이션 방법을 분석한 결과 Case 기반의 소스 코드를 재사용 하기위한 효율적인 의미망 구축을 위한 요구사항은 다음과 같다[5][8].

- 소스 코드를 분석하여 각 클래스 정보와 계층구조를 이루는 상속관계 정보의 추출기능을 제공해야 한다.
- 객체지향 컴포넌트의 특징인 상속성을 잘 표현할 수 있는 관계와 구조를 제공해야 한다.
- Case으로부터 관계성을 추출하고 추출된 관계성을 시스템 차원에서 파악해줌으로써 시소러스를 자동화할 수 있도록 해야 한다.
- 우리의 직감에 일치하는 잘 정의된 관계성을 포함한 시소러스를 도메인 전문가가 체계적이고 구조적으로 구축할 수 있도록 지원해야 한다.
- 구축과정에서 Case 내의 관계를 구조적으로 파악해서 구축자에게 필요한 정보를 제공함으로써 구축자의 부담을 최소화시켜야 한다.
- 클래스와 컴포넌트에 대한 시소러스 확장과 유지가 용이해야 한다.
- 스프레딩 엑티베이션의 장점은 유지하면서 빠른 검색을 위한 검색 메커니즘을 선정해야 한다.
- 시스템의 평가를 위한 정확도, 재현율 측정이 필요하다.

본 연구는 소스코드 기반 재사용을 위해 객체 지향 시소러스를 기반으로 한 의미망의 구축과 이를 통한 효율적인 검색을 위하여 스프레딩 엑티베이션 방법의 개선 방법을 제안한다. [그림 3]은 의미망 구축 방법을 나타낸 것이다.

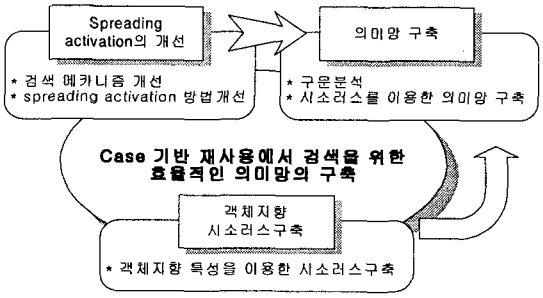

그림 3. 의미망의 구축 방법

3. 객체지향 시소러스 기반의 의미망 구축

본 연구에서 제안하는 의미망의 구축은 객체지향 시소러스를 기반으로 한다[6][7]. 이는 객체지향 코드에서 서로 관련 있는 클래스들을 일정한 기준에 따라 그룹화해서 여러 개의 클래스 그룹으로 구성하는 방식이다. 여기에서 여러 개의 클래스 그룹은 개념들 간에 나타나는 일종의 시소러스를 의미하며, 클래스의 상속관계를 이용하여 개념들 사이의 관계를 자연스럽게 표현해야 한다. [그림 4]는 소스 코드로부터 시소러스가 구축되는 전반적인 과정을 나타낸 것이다.

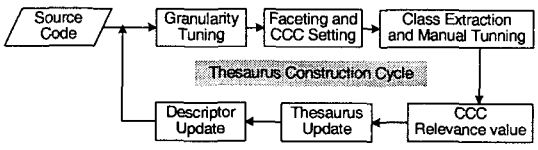

그림 4. 객체지향 시소러스 구축

다음은 본 연구에서 제안한 객체지향 시소러스를 기반으로 의미망을 구축하는 단계이다.

(1) 1 단계 : 클래스의 개념적 분류

클래스를 기능과 개념에 따라 여러 그룹으로 나눈다. 이 과정은 도메인 전문가에 의해 행해지며 시스템의 응용에 따라 모든 클래스를 최적으로 표현할 수 있는 개념을 선정한다. 각 개념은 패싯 항목으로 표현되고 이에 따라 클래스가 분류되어진다.

(2) 2 단계 : 클래스 범주 생성

개념적으로 분류된 클래스를 이용하여 클래스 개념 범주를 생성한다. 클래스가 사용될 수 있는 여러 경험적 상황을 패싯 항목으로 설정하는 패싯 분류방법을 사용한다. 클래스의 사용범위가 한가지로 국한되지 않고 여러 경우가 가능하며 이에 따른 경험적 솔루션을 제시해준다는 점에서 컴포넌트의 분류와 검색을 위해 클래스를 이용한 패싯 분류는 효율적인 방법이다.

(3) 3 단계 : 클래스와 범주의 관계값 계산

분류된 범주를 사용하여 클래스와 범주간의 관계값을 계산한다. 이 과정은 범주를 이용한 클래스 빈도를 계산하는 과정이다. 이 빈도값이 범주별 클래스 관계값이 되며, 이 관계값에 의해 초기 시소러스 유의어 테이블이 구성된다.

(4) 4 단계 : 객체지향 시소러스 생성

클래스와 범주의 관계값을 이용하여 퍼지 시소러스를 구축한다. 각 클래스와 범주에 대한 매칭 정도를 비교함으로써 이들 사이의 퍼지 정도를 계산하여 시소러스를 구축한다.

(5) 5 단계 : 시소러스 기반의 의미망 구축

소스 코드로부터 추출된 각 클래스와 메소드는 의미망의 노드와 간선으로 표현되며, 초기 각 노드의 관련값은 시소러스의 유의값으로 설정된다.

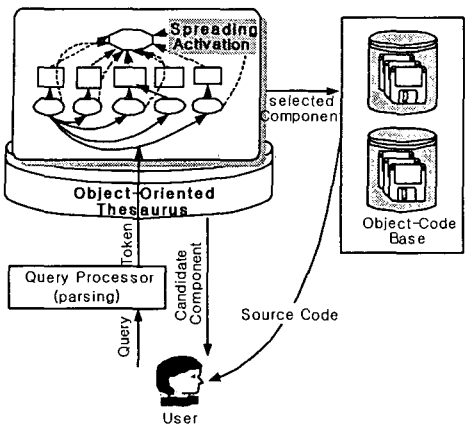

그림 5. 의미망을 이용한 컴포넌트 검색

Ⅳ. 검색방법의 개선

1. 의미망을 이용한 컴포넌트 검색

[그림 5]는 의미망을 이용한 컴포넌트 검색방법을 나타낸 것이다. 객체 지향 소스 코드를 재사용하기 위해 의미망을 구현하고 기능적으로 유사하거나 관련이 깊은 컴포넌트를 검색하는데 목적이 있다. 사용자는 컴포넌트의 행위적 특성을 입력하거나, 작업 중인 소스 코드를 질의로 입력할 수 있다. 입력된 질의는 질의어 처리기를 통하여 구문 분석되어 각 Case 별 의미망과 비교된다. 의미망에 있는 클래스와 컴포넌트는 객체지향 시소러스에 의해 초기 관련값이 설정되어 있고, 파싱된 질의는 의미망에서 매칭되는 각 노드를 활성화시킨다. 의미망의 노드와 간선의 초기 활성값을 이용하여 서로 연결되어 있는 노드를 참조해 가면서 활성값을 계산하게 되고, 순환이 반복될수록 활성값은 안정되며 참조회수가 기준에 미달되는 부분은 자동으로 제거되어 계산과정이 종료된다. 최종적으로 활성값이 가장 높은 컴포넌트들이 검색된다. 후보 컴포넌트는 순위에 따라 사용자에게 제공된다. 사용자가 이들 중 한 컴포넌트를 선택하고 선택된 컴포넌트의 소스 코드가 제공된다.

2. 검색방법의 개선

스프레딩 엑티베이션 알고리즘은 컴포넌트 간의 연관성에 대한 연결강도(connection relaxation)를 기초로 한 방법이다. 이 기술은 활성값이 안정되거나, 사용자가 설정한 최대 사이클 수만큼 실행하여 검색어에 연관이 있지만 직접 연결되지 않은 유사한 컴포넌트도 검색하도록 해준다. 이처럼 스프레딩 엑티베이션 알고리즘은 직접 인덱싱되어 있지 않은 컴포넌트까지 검색할 수 있는 효율적인 검색 방법이며 라이브러리에 컴포넌트들을 구축할 때 각 항목들을 일일이 인덱싱하지 않아도 되기 때문에 많은 비용이 절감된다. 그러나 이 방법은 검색할 때 활성값을 이용하여 유사도를 측정하기 때문에 시간이 많이 걸리는 단점이 있다. 이는 컴포넌트들이 증가할수록 활성값의 계산회수가 지수적으로 증가하기 때문에 컴포넌트들이 많을수록 검색에는 어려움이 발생한다[8]. 따라서 의미망을 효율적으로 구축하기 위해서는 스프레딩 엑티베이션의 성능은 최대한 유지하면서 검색 속도를 향상시킬 수 있는 방법이 요구된다.

이에 본 연구에서는 스프레딩 엑티베이션 방법의 단점을 해결하기 위하여 인덱싱이 적은 컴포넌트의 연결정보를 제거함으로써 활성값 계산회수를 줄여 검색시간을 단축시키는 방법으로 개선하고자 한다. 즉, 순환과정이 일정 수준 반복된 후 기준에 미치지 못하는 질의어나 컴포넌트의 연결정보를 제거하여 연산에서 제외시킴으로써 질의어의 확장범위를 줄여 보다 관계가 깊은 컴포넌트만을 검색하도록 하는 것이다.

제안한 스프레딩 엑티베이션 방법의 개선 방안은 다음과 같다.

- 1 단계 : 의미망의 각 노드에 시소러스의 유의값을 초기 활성값으로 설정.
- 2 단계 : 질의와 매칭되는 의미망 노드 활성화.
- 3 단계 : 활성화된 노드의 초기 활성값이 연결된 다른 노드로 이동.
- 4 단계 : 순환 반복.
- 5 단계: 노드 간 연결의 참조 회수를 이용하여 제거 기준 설정.
- 6 단계 : 참조회수가 기준에 못 미치면 연결삭제.
- 7 단계 : 순환이 끝나면 질의와 직접 연결된 컴포넌트의 활성값과 유사한 활성값을 갖는 컴포넌트 검색.

이를 알고리즘으로 나타내면 다음과 같다.

```
while
 Get_Level = Get_Pos[0] / End1;
            /* position of row matrix */
 Get_Col = Get_Pos[0] % End1;
          /* position of column matrix */
 if(exist component)
   for(all component number)
      Array[] = each component value(0 or 1)
 else
   for(all query number)
      Array[]= each query value(0 or 1)
 for(query or component number)
   if(exist relationship and index is Not last_index)
       push Current_index in Stack
 query_visit_count ++
 component_visit_count ++
   if(first query)
       initialize query_act_value=1.0
   else if(a query)
Di(t+1)=
δ_Di(t)+Ψ_Di(t)(M−D_i(t))  if Ψ_Di(t)>0
δ_Di(t)+Ψ_Di(t)(D_i(t)−m)  if Ψ_Di(t)≤0
δ_Di=(1−Θ_D)D_i(t)
    else if(a component)
Tj(t+1)=
δ_Tj(t)+Ψ_Tj(t)(M−T_j(t))   if Ψ_Tj(t)>0
δ_Tj(t)+Ψ_Tj(t)(T_j(t)−m)   if Ψ_Tj(t)≤0
δ_Tj=(1−Θ_T)D_j(t)
     else Err("Not Calculate Activation Value")
 pop(Get_Pos);
 if( Not MAX_CYCLE )
 { cycle++;
  if(cycle > MAX_CYCLE) /*MAX_CYCLE=3~5*/
         break;
  if(cycle is between 2 & 3)
    for( all query and components)
      if(query_visit_count or component_visit_count
        = current_cycle_count)
      {  AvgVisit += VisitNum[];
         cnt++; }
    if(cnt==0) return 0;
    else AvgVisit /= cnt;
   }
  if(visit_count exist)
   for( all query and component)
     if(VisitNum <= AvgVisit)
     {
       for(j =0 ; j < End1; j++)
          level0_1[j][i%End1]=0;
       Cut_component_value = -999.0;
       // 제거된 노드의 초기화
     } endwhile
```

[표 1]은 개선된 스프레딩 액티베이션 검색 알고리즘 방법(Enhanced Spreading Activation Retrieval Method)에서 해당 알고리즘에 사용된 변수들에 대한 정의를 나타낸 것이다.

표 1. E-SARM 파라미터

기호	정 의
$\delta_{D_i}(t)$	질의어 감소 비율로 감소된 이전의 활성값
$\Psi_{D_i}(t)$	현재(t) 컴포넌트 i에게 들어오는 입력값
M	최대 활성값 = 1.0
$D_i(t)$	이전 단계의 누적 컴포넌트 활성값
Θ_D	컴포넌트 감소 비율 = 0.1
$\delta_{T_j}(t)$	컴포넌트 감소비율로 감소된 이전의 활성값
$\Psi_{T_j}(t)$	현재(t) 질의어 i에게 들어오는 입력값
m	최소 활성값 = −0.2
$T_j(t)$	이전 단계의 누적 질의어 활성값
Θ_T	질의어 감소 비율 = −0.2

V. 실험 결과

본 연구에서는 자바 소스 코드를 질의로 주어 프로그래머의 의도에 가장 적합한 컴포넌트를 검색해 주는 방법을 실험하였다. 소스 코드에서 식별자가 추출되고, 이 식별자가 이미 리포지터리에 구성되어 있는 컴포넌트 의미망을 활성화시킴으로써 가장 크게 활성화된 컴포넌트를 검색하게 된다. 본 연구에서는 100개의 컴포넌트를 사용하여 시뮬레이션하였다. 각 컴포넌트는 10줄에서 120줄 사이의 자바 소스로 구성이 되어 있으며, 모두 의미망으로 변화되어 저장하였다. 새로운 컴포넌트가 추가될 때는 파싱 단계를 거쳐 자동적으로 의미망을 생성함으로써 시스템의 자동화를 달성할 수 있도록 하였다. 100개 컴포넌트에 대한 의미망은 약 1400개의 노드와 2400개의 간선으로 구성되었다. 시스템은 자바로 구현하였으며, Java 1.3 버전에서 구동하였다.

표 2. 활성값 계산 회수

컴포넌트 크기	기존의 스프레딩 액티베이션(회수)	개선된 스프레딩 액티베이션(회수)
20	2208.09	1710.58
40	3512.7	2541.255
60	8265.78	3398.15333
80	11652.01	6836.5825
100	23195.428	14210.466

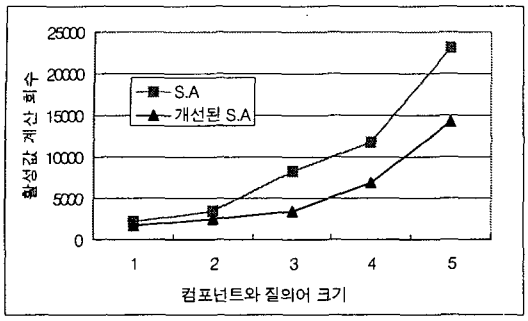

그림 6. 활성값 계산 회수 비교

[표 2]는 기존의 스프레딩 엑티베이션 방법과 본 논문에서 제안한 개선된 스프레딩 엑티베이션 방법을 컴포넌트와 질의어의 크기에 따라 활성값 계산 회수를 비교한 것이다. 여기서 제안한 스프레딩 엑티베이션 방법의 제거 기준은 평균 노드 참조회수의 88%로 설정하였다. 이는 활성값 계산 회수와 검색 결과는 서로 반비례하기 때문에, 검색 효율은 최대한 유지하면서 활성값 계산 회수를 최소한으로 할 수 있는 제거 기준을 설정한 것이다 [8]. [표 2]에 의하면 개선된 스프레딩 엑티베이션 방법이 기존의 스프레딩 엑티베이션 방법보다 활성값 계산 회수 면에서 평균 37.8% 감소되는 것을 볼 수 있다. 즉, 검색 시간이 현저히 단축됨을 의미한다. [그림 6]은 [표 2]를 그래프로 나타낸 것이다. 컴포넌트와 질의어의 크기가 커질수록 제안한 스프레딩 엑티베이션 방법이 더 좋은 효율을 보이는 것을 알 수 있다. [그림 7]은 본 실험에 대한 정확도를 측정한 것이다. 본 연구에서 제안한 의미망을 이용한 스프레딩 엑티베이션 기반 검색과 단순 시소러스 기반 검색방법을 비교하였다. 정확도 면에서 제안한 방법이 좋은 결과가 나타남을 알 수 있다.

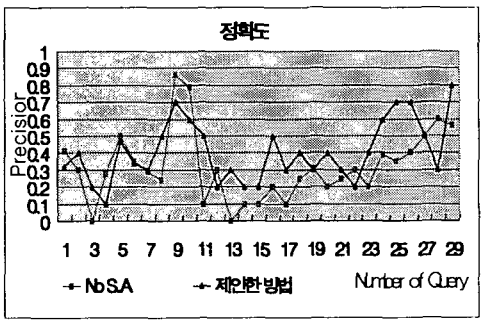

그림 7. 정확도

VI. 결 론

Case 기반 컴포넌트 검색을 위해 효율적인 의미망을 구축하기 위해서는 객체지향 파라다임을 검색에 적합한 형태로 해석·적용하고, 기존의 시소러스에서 이용된 관계성을 클래스와 컴포넌트의 관계로 재정의함으로써 의미망의 구축과 검색을 위한 새로운 기본 구조가 만들어져야 한다. 또한 구축된 의미망에 효율적인 검색 메커니즘을 적용함으로서 원하는 컴포넌트를 쉽게 검색할 수 있을 것이다. 따라서 본 연구에서는 의미망의 효율적인 구성을 위해서 각 노드 간 객체지향 상속의 개념을 표현할 수 있도록 의미망의 초기 관련값을 시소러스로 구축하였다. 또한, 의미망의 노드와 간선을 활성화시키고 활성값을 전파시키기 위해 사용되는 스프레딩 엑티베이션 방법의 단점을 보완하여 인덱싱이 적은 컴포넌트의 연결정보를 제거함으로써 활성값 계산회수를 줄여 검색시간을 단축시키는 방법을 제안하였다. 이를 통하여 객체 지향 소스 코드의 검색과 재사용을 효율적으로 수행할 수 있는 의미망을 구축하였다.

참 고 문 헌

[1] G. Markus and B. Derek, "Case-Based Reuse of Software Examplets," GWEM 2003 Proceedings of the Workshop on Experience Management, Apr., 2003.

[2] S. Henninger, "Information Access Tools for Software Reuse," System Software, pp.231-247, 1995.

[3] A. Aamodt and P. Plaza, "Case-Based Reasoning: Fundamental Issue, Methodological Variants, and System Approaches," Artificial Intelligence Communications, Vol.7, No.1, pp.39-59, 1994.

[4] P. Gomes, F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferriera, and C. Bento, "Using CBR for Automation of Software Design Patterns," Proceedings of the Sixth European

Workshop on Case-Based Reasoning, LNAI 2416, Springer, pp.543-548, 2002.

[5] G. J. Kim and J. S. Han, "Thesaurus Construction using Class Inheritance," Proceedings of the Computational Science and Its Application, LNCS 3482, Springer, pp.748-757, 2005.

[6] E. Damini, M. G. Fugini, and C. Bellettini, "A Hierarchy-Aware Approach to Faceted Classification of Object-Oriented Components," The ACM Transaction on Software Engineering and Methodology, Vol.8, No.4, pp.425-472, Oct., 1999.

[7] E. Damini and M. G. Fugini, "Automatic thesaurus construction supporting Fuzzy Retrieval of Reusable Components," In Proceedings of ACM SIG-APP Conference on Applied Computing, pp.542-547, Feb., 1995.

[8] 김귀정, 한정수, 송영재, "설계 패턴 기반 컴포넌트 분류와 E-SARM을 이용한 검색", 한국정보처리학회논문지,제11-D권, 제5호, pp.1133-1142, Dec., 2004.

저 자 소 개

한 정 수(Jung-Soo Han) 종신회원

- 1990년 : 경희대학교 전자계산공학과(공학사)
- 1992년 : 경희대학교 전자계산공학과(공학석사)
- 2000년 : 경희대학교 전자계산공학과(공학박사)
- 2001년~현재 : 백석대학교 정보통신학부 교수

<관심분야> : CBD, 컴포넌트 관리, CASE 도구