제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

의미망을 이용한 컴포넌트 기반 재사용

한정수*, 김귀정**
*천안대학교 정보통신학부, **건양대학교 컴퓨터공학과
e-mail:jshan@cheonan.ac.kr

Component-based Reuse using Semantic Network

Jung-Soo Han*, Gui-Jug Kim**
*Division of Info. & Comm. Cheonan University
**Dept of Computer Engineering, KonYang University

요 약

본 논문은 소프트웨어의 효율적인 재사용을 위해 소스 코드 기반 컴포넌트 검색 방법을 제안하였다. 제안한 방법은 2단계로 이루어지는데, 먼저 라이브러리에 저장된 클래스를 기반으로 한 컴포넌트는 파싱 과정을 거쳐 의미망을 구성하고, 다음으로 사용자가 질의한 소스 코드를 이용하여 검색이 이루어진다. 소스 코드에서 추출된 식별자가 컴포넌트의 의미망을 활성화시켜 연관된 컴포넌트를 검색한다. 본 연구에서 제안한 검색방법은 프로그래머의 관심을 라이브러리 내에 있는 컴포넌트로 유도하여 재사용성을 높일 수 있으며, 프로그래밍 패턴을 제공함으로써 프로그래머로 하여금 프로그램의 가이드 라인으로 사용할 수 있도록 도움을 줄 수 있다.

1. 서론

소프트웨어 개발이 진행되고 소프트웨어 산업이 발전하면서 가장 핵심이 되어왔던 부분 중 하나가 바로 컴포넌트 재사용 부분이다. 적절한 컴포넌트 재사용은 프로그래머의 개발 노력을 절감해 줄뿐만이 아니라 소프트웨어 품질을 향상시켜준다. 이에 따라 최근 20년 간 소프트웨어 재사용에 대해 많은 연구들이 이루어져 왔으며, 특히 재사용 컴포넌트 검색에 관해 다양한 노력이 이루어져 왔다.

본 논문은 소프트웨어의 효율적인 재사용을 위해 소스 코드 기반 컴포넌트 검색 방법을 제안하였다. 이 방법은 CBR(Case-Based Reasoning)을 기본으로 하여 검색(retrieval)-재사용(reuse)-수정(revise)-유지(retain)의 CBR 사이클에 따라 컴포넌트를 관리한다[1]. 제안한 방법의 검색은 크게 2단계로 이루어진다. 먼저, 라이브러리에 저장된 클래스를 기반으로 한 컴포넌트는 파싱 과정을 거쳐 의미망을 구성한다. 의미망은 소스 코드의 'class'등으로 구성된 노드와 'relevance'등으로 구성된 간선으로 만들어지며, 이 과정은 자동으로 이루어진다. 다음 단계는, 사용자가 질의로 준 자바 소스 코드를 이용한 검색 단계이다. 질의 소스 코드에서 추출된 식별자가 컴포넌트의 의미망을 활성화시켜 연관된 컴포넌트를 검색한다. 이때 의미망의 노드와 간선을 활성화시키고 활성값을 전파시키기 위해 스프레딩 엑티베이션 방법을 이용하였다. 최종적으로 가장 활성화가 많이 된 컴포넌트들이 검색된다. 본 연구에서의 컴포넌트 검색은 2가지 관점에서 재사용에 유용하다. 하나는 프로그래머의 관심을 라이브러리 내에 있는 컴포넌트로 유도하여 재사용성을 높일 수 있다. 이는 검색된 컴포넌트 내에 각 클래스를 하이퍼링크로 라이브러리 API 연결시키는 방법 등을 사용하여 재사용을 유도할 수 있다. 다른 하나는, 여러 다양한 클래스들이 포함된 전형적인 유형의 프로그래밍 패턴을 제공함으로써 프로그래머로 하여금 프로그램의 가이드 라인으로 사용할 수 있도록 도움을 준다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 컴포넌트 검색에 대해 설명하고, 3장에서는 의미망의 구성과 검색 방법에 대해 설명한다. 4장에서는 검색의 실험결과에 대해 설명하고, 끝으로 5장에

서 결론을 맺는다.

2. 관련 연구

소프트웨어 재사용에 대한 많은 연구들이 꾸준히 진행되고 있으며, 그중 컴포넌트 기반 검색에 대한 다양한 연구가 이루어지고 있다. 컴포넌트 기술하는 방법에도 여러 가지 방법이 있는데, 특성을 기본 요소로 하여 패싯 항목으로 표현하는 방법[2]이 있으며, 소프트웨어의 구조를 이용하여 의미망으로 표현하기도 하며, 클래스나 메소드등을 케이스 단위로 표현[3]하기도 한다. 이러한 컴포넌트 표현 방법에 따라 검색방법이 결정되어 진다. 컴포넌트 검색 연구에는 시그니쳐 일치 검색[4], 행위 샘플링에 의한 검색[5], 명세서 일치에 의한 검색[6] 등이 있다. 시그니쳐 일치 검색은 함수의 파라메터 타입이나 인터페이스와 같은 시그니쳐 정보를 이용하여 컴포넌트를 검색하는 방법이며, 행위 샘플링 검색은 인터페이스 명세서와 호환되는 인터페이스를 가진 저장소의 루틴을 실행하여 그 결과를 비교함으로써 검색하는 방법이다. 명세서 일치 방법은 컴포넌트 명세서를 비교하여 다른 컴포넌트와 대체될 수 있는지를 비교하여 결정한다. 스프레딩 엑티베이션(Spreading Activation Retrieval Method)[7]은 컴포넌트와 질의어 사이에 질의어 기능을 포함하는 유사한 컴포넌트들을 검색하여 보다 더 정확하고 넓은 범위의 컴포넌트들을 찾을 수 있는 방법이다. 이 방법은 직접 인덱싱되지 않은 컴포넌트들까지 검색할 수 있는 효율적인 검색 방법이며 정보저장소에 컴포넌트들을 구축할 때 각 항목을 일일이 인덱싱하지 않아도 되기 때문에 많은 비용이 절감된다.

3. 의미망에 의한 재사용

3.1 의미망의 구성

객체지향 언어와 같은 프로그래밍 언어는 다양한 종류의 재사용 컴포넌트를 사용한다. 이러한 라이브러리는 일반적으로 대단히 방대하다. 예를 들어, 자바의 경우 표준 클래스 라이브러리에 약 3000개의 클래스와 인터페이스가 포함되어 있다. 그러므로 컴포넌트 재사용을 효율적으로 하기 위해서는 방대한 컴포넌트 관리와 검색이 필요하다. 본 연구에서는 이러한 라이브러리에 포함되어 있는 자원을 프로그래머가 쉽게 사용할 수 있도록 필요한 컴포넌트를 효율적으로 검색하는 방법을 제안하고자 한다. 이 방법은 특히 라이브러리에 포함되어 있는 컴포넌트의 종류와 기능은 인지하고 있지만, 실제 프로그래밍에 적용하는 능력이 부족한 초급 프로그래머들에게 유용한 방법론을 제공해준다.

많은 CBR 시스템에서 라이브러리 내에 있는 각 클래스를 케이스(Case)로 취급하고 있으며, 또한 이러한 케이스를 재사용 단위인 컴포넌트화하고 있다. 본 연구에서는 컴포넌트를 라이브러리 내에 있는 클래스들, 또는 클래스 정의로 제한하여 사용하였다. 그림 1은 자바로 구현된 컴포넌트의 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. 본 연구에서는 단일 클래스 이름이나 자연어의 질의가 아닌 프로그래머가 작업하고 있는 자바의 소스 코드를 그대로 질의로 활용한다[3]. 이는 다양한 클래스들이 포함된 전형적인 유형의 프로그래밍 패턴을 제공함으로써 프로그래머로 하여금 프로그램의 가이드 라인으로 사용할 수 있도록 도움을 줄 수 있다.

```
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();
  }
}
```

그림 1. 'URL_Reader' 컴포넌트

라이브러리에 있는 각 컴포넌트는 파싱 단계를 거

처 파싱 트리를 형성한다. 본 연구에서는 'class', 'interface', 'method', 그리고 'variable'을 식별자로 추출하였다. 추출 과정은 컴포넌트 소스 코드를 읽어 각 식별자를 추출하고 식별자 사이의 관계 정보를 저장한다. 본 연구에서 제안한 의미망은 파싱 트리로부터 만들어진다. 파싱에서부터 생성된 'class', 'interface', 'method', 'variable'은 의미망에서 노드로 구성되며, 'relevance', 'subclass', 'implements', 'member', 그리고 'invoke'등은 클래스들 간의 관계로 취급하여 노드간 간선으로 구성하였다. 그림 1의 컴포넌트에 대한 의미망은 그림 2와 같이 구성된다.

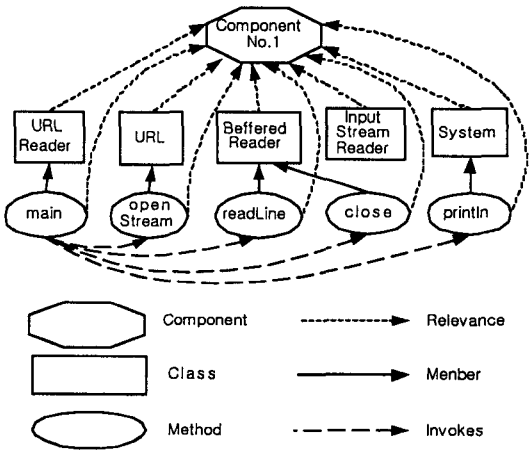

그림 2. 'URL_Reader' 컴포넌트의 의미망

3.2 의미망을 이용한 검색

본 연구는 컴포넌트의 소스 코드로부터 구성된 컴포넌트 의미망을 이용하여 컴포넌트를 검색한다. 컴포넌트 검색은 사용자에 의해 주워진 질의가 의미망을 활성화시킴으로써 이루어진다. 사용자는 자신이 작성하고 있는 자바 소스 코드를 질의로 사용한다. 이때, 사용자는 라이브러리에 있는 클래스 종류와 기능은 숙지하고 있으나, 문제를 해결하기 위해 어떤 방법을 이용해야 할지 모르는 프로그래머이다. 질의로 주워진 소스 코드는 구문 분석을 통해 식별자로 추출되는데, 프로그램 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[3]을 이용하였다. 소스 코드의 식별자는 각 컴포넌트의 의미망에 있는 노드와 비교되어 매치되는 노드를 활성화시킨다. 식별자 중 클래스는 이에 해당하는 의미망의 클래스 노드를 활성화시키고, 클래스 변수와 메소드는 이에 매치되는 의미망의 'variable' 노드와 'Method'를 활성화시킨다. 이때, 식별자의 수만큼 반복하여 해당 노드들이 활성화된다. 각 노드의 초기값은 질의로 주워진 소스 코드의 식별자와 의미망에서 매치되는 노드 간의 유사도로 설정되며, 0과 1사이의 값을 갖는다. 노드간의 관계를 나타내는 의미망의 간선은 초기 활성값 1을 갖는다. 이 초기값은 소스 코드에서 메소드 호출 등과 같이 관계가 반복적으로 나타날 때마다 증가하게 된다. 본 연구에서는 증가치를 1.2로 설정하였다.

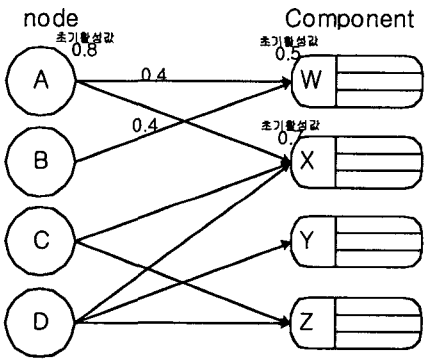

그림 3. 활성값 전파 과정

컴포넌트의 검색은 의미망을 통한 스프레딩 엑티베이션에 의해 이루어진다. 의미망의 노드와 간선의 초기 활성값을 이용하여 서로 연결되어 있는 노드를 참조해 가면서 활성값을 계산하게 된다. 순환이 반복될수록 활성값은 안정되며 참조회수가 기준에 미달되는 부분은 자동으로 제거되어 계산과정이 종료된다. 그림 3은 스프레딩 엑티베이션에 의한 활성값 전파 과정을 설명한 것이다. 노드 A의 초기 활성값은 0.8이고, 컴포넌트 W의 초기 활성값은 0.5이다. 노드 A가 활성화되면 3개의 컴포넌트가 검색된다. 여기서 노드 A는 W와 X에 직접 연결되어 있지만 Y와 Z에는 연결되어 있지 않다. A→X→D→Z를 통하여 연결되고, A→X→D→Y를 통하여 2개의 컴포넌트(Z, Y)가 연결됨을 알 수 있다. 하지만 Y는 검색과정에서 적게 참조되므로 연결이 제거된다. 본 연구에서는 계산되는 활성값이 임계값 0.1 미만일 경우에 검색을 종료하도록 하였으며 최대 시행 횟수

를 150회로 설정하였다. 최종적으로 활성값이 가장 높은 컴포넌트들이 검색된다.

4. 실험 결과

본 연구는 자바 소스 코드를 질의로 주어 프로그래머의 의도에 가장 적합한 컴포넌트를 검색해 주는 방법을 제안하였다. 소스 코드에서 식별자가 추출되고, 이 식별자가 이미 리포지터리에 구성되어 있는 컴포넌트 의미망을 활성화시킴으로써 가장 크게 활성화된 컴포넌트를 검색하게 된다. 본 연구에서는 40개의 컴포넌트를 사용하여 시뮬레이션하였다. 각 컴포넌트는 10줄에서 120줄 사이의 자비 소스로 구성이 되어 있으며, 모두 의미망으로 변화되어 저장하였다. 새로운 컴포넌트가 추가될 때는 3.1에서 설명한 것과 같이 파싱 단계를 거쳐 자동적으로 의미망을 생성함으로써 시스템의 자동화를 달성할 수 있도록 하였다. 40개 컴포넌트에 대한 의미망은 약 340개의 노드와 480개의 간선으로 구성되었다. 시스템은 자바로 구현하였으며, Java 1.3 버전에서 구동하였다. 그림 4는 본 실험에 대한 정확도를 측정한 것이다. 본 연구에서 제안한 의미망을 이용한 스프레딩 엑티베이션을 적용한 검색방법과 단순 시소러스 기반 검색방법을 비교하였다. 정확도 면에서 제안한 방법이 좋은 결과가 나타남을 알 수 있다.

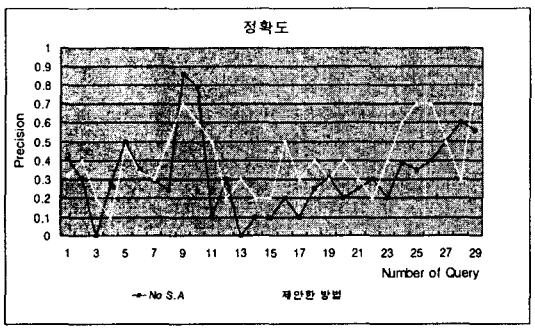

그림 4. 정확도

5. 결론

본 연구는 효율적인 소프트웨어 재사용을 지원하기 위하여 의미망을 이용한 컴포넌트 검색방법을 제안하였다. 의미망은 라이브러리에 있는 컴포넌트를 구문 분석하여 노드와 간선으로 표현하였고, 이는 사용자가 질의로 준 소스 코드에 의해 활성화되어 소스 코드와 가장 연관성이 높은 컴포넌트가 검색된다. 의미망을 활성화시키기 위한 활성값 계산은 스프레딩 엑티베이션 방법을 이용함으로써 각 노드의 참조 횟수와 활성값에 따라 컴포넌트 검색이 이루어지도록 하였다. 의미망의 구성과 망 전파에 의한 검색 과정은 구문 분석에 의해 자동으로 이루어지며, 이는 프로그래머의 작업에 많은 도움을 준다. 따라서 본 논문에서 제안한 의미망을 이용한 검색방법은 프로그래머의 요구에 능동적으로 반응하여 재사용에 최적의 컴포넌트를 제공해 준다.

향후 연구는 질의로 주워진 소스 코드에서 추출한 식별자와 의미망에서 노드와의 유사성을 좀더 정확히 파악하는데 있다. 또한 좀더 다양한 컴포넌트에의 적용이 필요할 것이다.

참고문헌

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

[2] Prieto Diaz, R, Freeman, P, "Classification Software for Reusability," IEEE Software, Vol. 4, No. 1, 1987. 6-16.

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

[4] A. M. Zaremski, J. M. Wing, "Signature Matching: A Tool for Using Software Libraries," ACM Transaction Software Engineering and Methodology, Vol. 4, No. 2, 1995.

[5] A. Podgurski, L. Pierce, "Retrieving Reusable Software by Sampling Behavior," ACM Transaction Software Engineering and Methodology, Vol. 2, No. 3, 1993.

[6] A. M. Zaremski, J. M. Wing, "Specification Matching of Software Components," In Proceedings of the third ACM SIGSOFT symposium on the foundations of software engineering, 1995.

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