소프트웨어 아키텍처 기반의 재사용 가능한 컴포넌트 검색

(Reusable Component Retrieval Based on Software Architecture)

이 승 근 † 안 치 돈 †† 이 윤 수 ††† 왕 창 종 †††

(Seung Geun Lee)(Chi Don Ahn)(Yoon Soo Lee)(Chang Jong Wang)

요 약 이 연구에서는 재사용을 위한 소프트웨어 아키텍처 검색 방법과 아키텍처의 재사용에 필요한 컴포넌트를 효율적으로 검색하기 위해 변경된 시그니처 일치와 행위 일치 검색 방법을 제안한다. 제안된 검색 방식은 소프트웨어 아키텍처 검색을 통해 컴포넌트의 검색 범위를 제한함으로써 검색의 정확성과 재현율을 향상시키고, 일치 수행의 비교 횟수를 줄임으로써 검색 수행 시간을 감소시켰다. 또한 이미 정의된 소프트웨어 아키텍처와 컴포넌트의 재사용성을 향상시킬 수 있다.

Abstract *This paper proposes a software architecture retrieval method for reuse. In addition to this, it also proposes modified signature match and behavior match to retrieve components efficiently for architecture reuse. In proposed retrieval methods, software architecture retrieval improves the recall and precision of retrieval through restricting the retrieval scope of a component. It also reduces overall retrieval execution time by decreasing the number of comparison for matching. In addition to these, it improves reusability of predefined software architectures and components.*

1. 서 론

소프트웨어 공학에서 소프트웨어 재사용(software reuse)은 소프트웨어 위기 현상을 극복하기 위해 현재 널리 연구되고 있는 분야이다[1]. 소프트웨어 재사용은 이미 개발된 컴포넌트(components)를 새로운 소프트웨어 개발에 사용함으로써 소프트웨어 개발 기간을 단축시키고, 개발 비용의 절감과 소프트웨어의 생산성을 향상시킬 수 있다[1, 2].

기존의 대표적인 컴포넌트 검색 방법에는 파라미터 타입이나 인터페이스와 같은 시그니처 정보를 사용하는 시그니처 일치(signature match) 방법[3]과 소프트웨어의 실행 속성을 기반으로 하는 행위 샘플링(behavioral sampling) 방법[4], 두 개의 컴포넌트 명세서를 비교하는 명세서 일치(specification match) 방법[5]이 있다. 기존의 검색 방법들은 저장소에 저장된 전체 컴포넌트를 대상으로 하므로 검색 시간이 길고, 컴포넌트의 함수 타입이나 기능적인 측면만을 비교하기 때문에 검색 효율과 검색의 정확성이 요구된다.

따라서, 이 연구에서는 소프트웨어 아키텍처 명세(software architecture specification)와 컴포넌트 명세(component specification)가 저장소에 저장되어 있을 때, 개발자의 질의와 구조적으로 일치하는 아키텍처를 검색하는 아키텍처 검색 방법과, 검색된 아키텍처를 구성하는 컴포넌트를 검색하기 위한 변경된 시그니처 일치 방법과 변경된 행위 일치 방법을 제안한다. 제안된 방식을 사용하여 새로운 소프트웨어 어플리케이션을 개발에 필요한 적절한 아키텍처와 컴포넌트를 검색하는 검색 시스템을 프로토타이핑(prototyping) 개발 방법에 적용하여 전체적인 검색 구조를 설계하고 구현함으로써 현재 사용자 상호 작용을 지원하는 GUI 응용 개발에 적용할 수 있도록 한다.

· 이 논문은 2000학년도 안산공과대학 학술연구비에 의해서 이루어졌음.

† 학생회원 : 인하대학교 전자계산공학과
sglee@selab.cse.inha.ac.kr
†† 비 회 원 인하대학교 전자계산공학과
joheunid@netsgo.com
††† 정 회 원 : 안산공과대학 전산정보과 교수
yslee@ansantc.ac.kr
†††† 종신회원 : 인하대학교 전자계산공학과 교수
cjwang@dragon.inha.ac.kr

논문접수 : 1999년 11월 17일
심사완료 : 2000년 9월 25일

아키텍쳐 검색은 소프트웨어의 구조적인 측면에서 개발자의 요구 사항과 일치하는 아키텍쳐를 검색하고, 검색된 아키텍쳐를 구성하기 위한 컴포넌트의 검색 범위를 축소시킴으로써, 컴포넌트 검색의 재현율(recall)과 정확성(precision)을 향상시킬 수 있고, 검색 수행 시간 또한 감소시킬 수 있었다.

2. 소프트웨어 아키텍쳐와 컴포넌트 검색

이 장에서는 소프트웨어 아키텍쳐의 기본 개념과 기존의 컴포넌트 검색 방법, 그리고 검색 효율을 평가하는 기준에 대해 고찰한다.

2.1 소프트웨어 아키텍쳐

최근 소프트웨어 공학에서의 연구 경향은 모듈 인터페이스 언어(module interface language), 특정 영역 아키텍쳐(specific domain architecture), 구조적 기술 언어(architectural description language), 설계 패턴(design pattern), 구조적 설계(architectural design)를 위한 기반, 그리고 구조적 설계 환경(architectural design environment) 영역에 대해 활발한 연구가 이루어지고 있다. 소프트웨어 아키텍쳐는 프로그램과 시스템 컴포넌트, 그들의 상호관계에 대한 구조, 그리고 컴포넌트를 설계하고 개발하기 위한 규칙 및 지침서라고 정의한다[6].

소프트웨어 아키텍쳐와 관련한 최근 두 가지 경향은 복잡한 소프트웨어 시스템을 구조화하기 위해 메소드, 기술, 패턴, 특징(idioms) 등을 공유하는 것과 사용 가능한 프레임워크를 제공하기 위한 특정 영역의 이용 문제에 관한 것이다[7]. 소프트웨어 개발에서 규칙화 된 소프트웨어 구조를 사용하게 되면 시스템 설계에서 추상화 레벨로 시스템을 표현함으로 복잡한 시스템에 대한 이해를 돕고, 컴포넌트 라이브러리를 통해 다중 레벨에서 재사용이 가능하다. 그리고 시스템의 발전 방향 예측이 가능하고, 소프트웨어 유지 보수자로 하여금 변화를 보다 쉽게 이해할 수 있도록 함으로써 수정에 대한 비용을 정확하게 측정할 수 있게 하며, 시스템의 일관성 검사, 구조적 스타일의 일치, 특정 영역 분석을 위한 고수준의 형식을 제공한다.

2.2 기존의 아키텍쳐 명세 언어

메시지 기반의 소프트웨어 아키텍쳐 명세 언어로는 Rapide와 C2 등이 있다. Rapide는 분산 시스템 구조의 프로토타이핑(prototyping)을 위해 설계된 이벤트 기반의 동시 객체 지향 언어(event-based concurrent object-oriented language)이다. Rapide에서 아키텍쳐는 모듈에 대한 명세 집합, 인터페이스간 직접적인 통신을 정의하는 연결 규칙의 집합, 그리고 통신 패턴을 정의하는 정형화된 제약조건으로 구성된다[8].

C2는 메시지 기반 스타일을 사용하여 사용자 인터페이스를 기술하는 언어이다. C2의 구조적 스타일에서 주된 두 가지 요소는 아키텍쳐의 구성 요소인 컴포넌트와 커넥터(connectors)이다. 컴포넌트는 커넥터를 통해서만 통신할 수 있으며, 하나의 커넥터에 연결될 수 있는 컴포넌트, 또는 커넥터의 수에는 제한이 없다. C2에서 컴포넌트간 통신은 메시지 전달을 통해 이루어지며, 커넥터는 컴포넌트에서 발생한 이벤트를 다른 컴포넌트에 전달한다[9].

대부분의 명세 언어들은 기능에 관한 명세로 비기능의 명세 지원이 미흡하고, 특정 언어에 종속적이며, 명세되는 시스템의 관점이 제한적이다.

2.3 기존의 컵포넌트 검색 방법

기존의 컴포넌트 검색 방법에는 시그니쳐 일치에 의한 방법과 행위 샘플링에 의한 방법, 그리고 명세서 일치에 의한 방법이 있다.

시그니쳐 일치 검색 방법은 컴포넌트로부터 얻을 수 있는 함수의 파라미터 타입이나 인터페이스와 같은 시그니쳐 정보를 이용하여 컴포넌트를 결정하는 방법으로 함수의 타입과 질의의 타입이 정확히 일치하는 컴포넌트를 검색하는 완전 일치(exact match) 방법과 유사한 컴포넌트들을 반환하는 이완 일치(relaxed match) 방법이 있다[3].

행위 샘플링 검색 방법은 대상 인터페이스 명세서와 호환되는 인터페이스를 가진 저장소의 루틴을 임의로 선택된 연산 입력의 샘플 상에 실행시키고, 임의로 선택된 출력과 루틴의 실행 결과를 비교함으로써 컴포넌트를 검색하는 방법으로 검색의 자동화를 위해 제안되었다[4].

명세서 일치 방법은 두 개의 컴포넌트 명세서를 비교하기 위한 방법으로, 하나의 컴포넌트가 다른 컴포넌트에 대체될 수 있는지를 결정할 수 있도록 한다[5]. 컴포넌트 명세서와 질의 명세서의 관련성을 평가하는 기준에 따라 각 컴포넌트의 선조건과 후조건을 비교하는 선조건/후조건 일치(pre/post match)와 술어 포함 관계를 이용하여 컴포넌트를 검색하는 술어 일치(predicate match) 방법 등이 존재한다[5].

2.4 검색 효율 평가 기준

새로운 소프트웨어 개발에 소프트웨어 컴포넌트를 사용하기 위해서 많은 검색 방법들이 제안되었다. 소프트웨어 컴포넌트 검색 방법은 일반적으로 정확성, 재현율, 시간 복잡성, 그리고 자동화라는 평가 기준들을 이용하

여 검색 효율을 평가한다[10, 11].

정확성은 검색된 모든 컴포넌트 중에 사용자의 요구사항을 만족하는 컴포넌트의 비율을 나타낸다. 재현율은 저장소에 저장된 모든 관련 컴포넌트 중 사용자의 요구사항과 관련된 컴포넌트가 얼마나 검색되었는가를 나타낸다. 시간 복잡성은 질의와 컴포넌트를 일치시키기 위해 요구되는 연산 단계의 수를 나타낸다. 마지막으로 자동화는 검색 방법을 자동화할 수 있는지를 나타낸다.

정확성과 재현율은 일반적으로 반비례 형태로 나타난다[10]. 컴포넌트 검색에서 개발자가 요구하는 컴포넌트가 모두 검색되고, 검색된 컴포넌트가 모두 관련 컴포넌트라면 가장 이상적인 방법이라 할 수 있지만 실제 구현은 쉽지 가 않다. 따라서, 정확성과 재현율을 최대한 높일 수 있는 방법이 고려되어야 한다[10, 11]. 시간 복잡성은 질의의 크기에 따라 시간 비용을 측정하기 때문에, 질의 작성을 최소화하여야 한다. 하지만, 질의 작성이 너무 간단하면 검색의 정확성과 재현율이 낮아지는 단점이 있다[10]. 따라서, 검색 방법들은 시간 비용을 최소화하고, 정확성과 재현율을 최적화할 수 있어야 한다.

3. 아키텍쳐 기반 컴포넌트 검색 시스템

이 장에서는 소프트웨어 아키텍쳐 저장소로부터 아키텍쳐를 검색하고, 검색된 아키텍쳐에 포함되는 컴포넌트를 검색함으로써 개별 컴포넌트 검색의 효율을 향상시킬 수 있는 방안을 제안한다.

3.1 아키텍쳐 검색 방법

아키텍쳐 검색은 개발자의 질의와 구조적으로 일치하는 아키텍쳐를 검색한다. 소프트웨어 아키텍쳐 저장소에 저장된 아키텍쳐들은 다수의 컴포넌트와 커넥터로 구성되어 있다. 개발자의 질의와 일치하는 아키텍쳐를 검색을 위해서는 소프트웨어 아키텍쳐 내의 이러한 컴포넌트와 커넥터를 일치시켜야 한다.

이 연구에서는 아키텍쳐 내에 존재하는 모든 컴포넌트의 수를 검사하고, 컴포넌트에 의해 발생되는 메시지를 다른 컴포넌트로 전달하는 커넥터의 수를 비교하여 아키텍쳐를 검색한다. 검색된 아키텍쳐는 소프트웨어 아키텍쳐의 재구성을 위해 사용된다. 알고리즘 1은 아키텍쳐 검색을 위한 알고리즘이다.

알고리즘 1 아키텍쳐 검색 알고리즘

```
Query Architecture : Q
Architecture Repository : AR
Candidate Architectures : C
Begin Architecture Match
 Arch a := AR.getFirstArch(); flag = FIND_OK;
 While (a != NULL)
  If(a.getCompCount()==Q.getCompCount()) Then
   If (a.getConnCount)>=Q.getConnCount()) Then
    CompList QueryComps := Q.getSimpComps();
    CompList ArchComps := a.getSimpComps();
    ConnList ArchConnList := a.getConns();
    For (i = 0; i < QueryComps.getSize(); i++)
     SelectedComp := ArchComps.findWithName(
                          QueryComps[i].name);
     If (SelectedComp == NULL) Then
      flag := MATCH_FAIL; break; End If
     MsgList CompMsgList :=
                          QueryComps[i].getMsgs();
     MsgList SCMsgList :=
                       SelectedComps[i].getMsgs();
     For (j=0; j<CompMsgList.getSize(); j++)
      SelectedConn := SCMsgList.findWithReturn(
                          CompMsgList[j].return);
      If (SelectedConn != NULL) Then
       If(SelectedConn.source !=
                       CompMsgList[i].source) Then
        flag := MATCH_FAIL; break; End If
       If (SelectedConn.target !=
                       CompMsgList[i].target) Then
        flag := MATCH_FAIL; break; End If
      End If
     End For
     If (flag == MATCH_FAIL) Then break;
    End For
   End If
  End If
  If(flag!=MATCH_FAIL) Then C.addArchitecture(a);
  a := AR.getNextArch();
 End While
End Architecture Match
```

이 알고리즘에서 아키텍쳐의 컴포넌트 수와 질의의 컴포넌트 수를 비교하여 일치하지 않으면 검색 대상에서 제외된다. 다음 과정으로 일치한 컴포넌트에 대해 커넥터 수를 비교한다. 비교 결과 많거나 동일한 개수의 커넥터를 포함하고 있는 아키텍쳐에 대해 컴포넌트 명 및 커넥터의 반환 타입과 소스 및 목표 컴포넌트 정보를 비교하여 일치하는 아키텍쳐를 검색한다.

검색된 아키텍쳐가 실제 소프트웨어 개발에 적합한 아키텍쳐가 아닐 경우는 아키텍쳐 재구성에 적합한 컴

포넌트를 검색한 다음, 후보 아키텍쳐(candidate architecture)의 컴포넌트를 검색된 컴포넌트로 대체하여 새로운 소프트웨어 아키텍쳐를 재구성하여야 한다.

3.2 컴포넌트 검색

이 절에서는 컴포넌트 검색을 위한 변경된 시그니쳐 일치 검색(modified signature match)과 행위 일치 검색(modified behavior match) 방법을 제안한다.

(1) 시그니쳐 일치 방법에 의한 검색

시그니쳐 일치 방법은 컴포넌트의 파라미터 타입과 파라미터의 입출력 방향을 개발자의 질의와 비교하여 컴포넌트를 검색하는 방법이다. 변경된 시그니쳐 일치 방법의 일반적인 일치 술어는 다음과 같이 정의한다.

$M(L, Q) = (D_l, R_1, D_q) \wedge (T_l, R_2, T_q)$

T_l과 T_q는 컴포넌트 인터페이스와 질의의 파라미터 타입들이고, D_l 과 D_q는 컴포넌트 인터페이스의 메소드와 질의의 파라미터 입출력 방향을 나타내며, R_1과 R_2는 컴포넌트의 메소드와 질의와의 관련성을 나타낸다. 이 관련성에 따라 완전 일치, 타입 이완 일치, 방향 이완 일치 방법이 있다. 각 방향의 파리미터 개수와 질의의 파라미터 개수를 비교하여 일치하면 완전 일치 방법이나 타입 이완 일치 방법을 수행한다. 만약 일치하지 않을 경우 방향 이완 일치 방법을 수행한다.

완전 일치 방법은 컴포넌트의 입출력 방향의 수와 질의의 입출력 방향의 수가 같을 때, 각 방향에 대한 타입을 비교하여 컴포넌트를 검색한다. 타입이 일치하지 않을 경우에는 검색 대상에서 제외된다. 이러한 컴포넌트 검색은 정확한 컴포넌트를 검색할 수는 있지만 소프트웨어 개발에 이용될 수 있는 유사한 컴포넌트는 검색할 수 없다.

타입 이완 일치 방법은 명세된 컴포넌트들을 질의와 비교할 때, 입출력 방향의 수는 반드시 고려되어야 하고, 입출력 방향의 타입이 질의와 다른 컴포넌트들을 검색하여야 한다. 따라서 질의나 컴포넌트의 파라미터 입출력 방향의 수가 일치하였을 때, 파라미터 타입만을 비교하여 컴포넌트를 검색한다.

방향 이완 일치 방법은 컴포넌트 명세에서 파라미터의 입출력 방향을 나타내는 입출력(inout) 파라미터가 입력 방향과 출력 방향 모두에 적용될 수 있기 때문에, 이와 같은 경우에 질의와 일치하는 컴포넌트를 검색한다. 이 방법은 컴포넌트나 질의의 파라미터 입출력 방향이 수가 일치하지 않을 때, 입출력 방향의 수를 조정하여 일치시킨 후, 각 방향의 타입을 검사하여 질의와 일치하는 컴포넌트를 검색한다. 알고리즘 2는 방향 이완 일치 검색 알고리즘을 나타내고 있다.

알고리즘 2 방향 이완 일치 검색 알고리즘

```
Component: C[], Query: Q
int i := j := 0;
 InTemp := Incount - rIncount;
 outTemp := Outcount - rOutcount;
 inoutTemp := inoutcount - rinoutcount;
 If((inTemp == ountTemp) &&
   (inTemp == (-inoutTemp)) then C[i++] := R;
 While(C[j] == NULL)
   if(!checkType(C[j].inType, Q.inType)) then
     j++; continue; End if
   if(!checkType(C[j].outType, Q.outType)) then
     j++; continue; End if
   if(!checkType(C[j].inoutType, Q.inoutType)) then
     j++; continue; End if
   if(compareType(C[j].inType, C[j].outType)) then
     if(compareType(C[j].inType,
                C[j].inoutType)) then return(true);
  End while
End Direction Relaxed Match
```

(2) 변경된 행위 일치에 의한 검색

행위 일치 방법에 의한 검색에서 컴포넌트 저장소에 있는 모든 컴포넌트를 대상으로 하는 것은 검색 효율을 감소시키므로, 변경된 시그니쳐 일치 방법에 의해 검색된 컴포넌트를 대상으로 하여 개발자의 요구와 일치하는 컴포넌트를 검색한다. 개발자는 컴포넌트 인터페이스를 시뮬레이션하기 위한 입력값과 시뮬레이션 결과와 비교하기 위한 출력값을 설정하고, 이를 비교하여 적절한 컴포넌트를 검색한다.

그림 1은 변경된 행위 일치에 의한 검색 방법의 구조를 보여주고 있다. 컴포넌트 저장소의 컴포넌트는 가상 머신(virtual machine) 상에서 개발자가 입력한 값을 가지고 실행된다. 평가기(evaluator)는 실행 결과와 개발자가 설정한 출력값을 비교하여 개발자의 요구 사항과 일치하는 컴포넌트를 검색한다. 변경된 행위 일치 검색 방법을 사용하더라도 다수의 컴포넌트가 검색될 수 있고, 개발자의 요구 사항과 정확하게 일치하지 않는 컴포넌트들이 검색될 수 있기 때문에 개발자는 적절한 커넥터를 사용하여 소프트웨어 개발에 적용하여야 한다.

변경된 행위 일치 방법은 변경된 시그니쳐 일치 검색에 의해 검색된 컴포넌트를 대상으로 검색을 수행하므로 검색 시간 비용을 줄일 수 있고, 컴포넌트 사용자가 직접 입력값과 출력값을 설정하기 때문에 검색의 재현

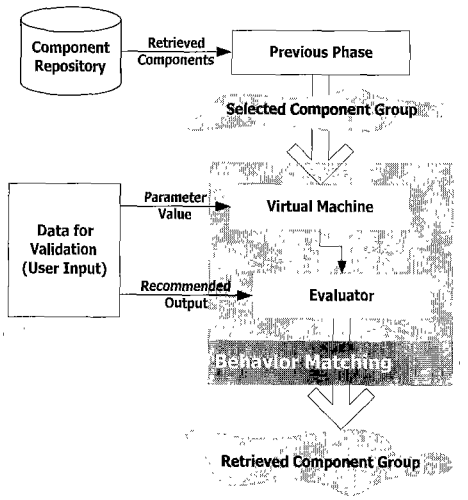

그림 1 변경된 행위 일치 검색 구조

율을 향상시킬 수 있다.

4. 실험 및 평가

이 장에서는 제안한 아키텍처 검색 방법을 이용하여 아키텍처를 검색하고, 컴포넌트 검색 방법의 검색 효율을 평가하기 위해 정확성과 재현율을 측정하였다.

아키텍처 검색을 위한 실험은 사용자 인터페이스 아키텍처를 재구성하기 위해 사용자 인터페이스를 대상으로 하여 질의와 일치하는 아키텍처를 검색한다. 사용자 질의가 주어지면 아키텍처 검색 알고리즘에 따라 아키텍처 저장소로부터 사용자 인터페이스 아키텍처를 검색하여 후보 아키텍처를 선택하게 되고, 선택된 아키텍처들을 재구성하기 위하여 컴포넌트 검색을 수행하게 된다. 컴포넌트 검색을 위해 사용자 질의를 재구성하여 각 컴포넌트에 해당되는 질의를 생성하고, 생성된 질의는 시그니처 일치 검색 방법과 행위 일치 검색 방법을 사용하여 컴포넌트 저장소로부터 적합한 컴포넌트를 검색한다. 정확성과 재현율은 아키텍처 검색을 수행한 후, 아키텍처에 필요한 컴포넌트를 검색한 결과와 컴포넌트 검색만을 수행한 결과를 비교하여 측정한다.

프로토타입 검색 시스템의 개발 환경은 Microsoft Windows NT 4.0 Server, Microsoft Transaction Server에서 Microsoft Visual C++ 6.0을 사용하였고, XML 문서 검색을 위한 검색 엔진으로는 IXIA soft사의 TEXTML/SERVER Lite를 사용하였다. 검색 엔진을 사용하기 위해 컴포넌트 클라이언트 형태의 컨트롤을 구현하여 사용하였으며, 사용자의 질의는 검색 서버의 컴포넌트에 전달되어 검색되고, 검색된 문서들을 클라이언트 컨트롤에서 비교, 조합하는 형식의 응용을 구성하였다.

실험에 사용한 컴포넌트 저장소는 Java 2 SDK 1.2 버전에서 제공하는 AWT (Abstracted Window Toolkit)와 Javax 패키지의 사용자 인터페이스 클래스 (User Interface Class) 728개 중에서 일부를 XML 기반 명세 언어로 정의하여 사용하였고, 사용자와의 상호작용에 관련된 클래스 구조들을 아키텍처로 정의하여 XML기반의 명세 언어로 표현하여 사용하였다 사용자 인터페이스 컴포넌트의 개수는 총 322개이며, 아키텍처 검색을 위해 일반적으로 사용되는 사용자 인터페이스 아키텍처 81개를 추가로 정의하여 아키텍처 저장소에 저장하였다. 표 1은 실험 수행 결과를 컴포넌트 검색만 수행한 결과와 아키텍처 검색을 병행한 경우로 구분하여 정리한 것이다.

그림 2는 실험 수행 결과에 따른 아키텍처 기반 검색 방법과 기존 컴포넌트 검색 방법의 재현율과 정확성을 그래프를 사용하여 비교한 것이다.

아키텍처 내에 컴포넌트 개수가 증가함에 따라 재현율과 정확성이 기존 방법보다 높은 것을 볼 수 있다. 이는 다량의 컴포넌트를 동시에 검색하는 경우, 각각의 컴포넌트를 단독으로 검색하는 것보다 소프트웨어 개발을 위해서 아키텍처 개념을 도입함으로써 구조 단위의 검색을 수행할 수 있도록 하는 것이 더 효율적임을 나타내고 있다.

표 1 실험 결과

Comps in Arch	Component Retrieval Only		With Architecture Retrieval	
	Recall	Precision	Recall	Precision
4	1	0.25	1	0.69
7	0.75	0.34	0.77	0.71
11	0.67	0.43	0.68	0.68
17	0.65	0.52	0.69	0.71
20	0.7	0.41	0.75	0.58
22	0.73	0.6	0.78	0.75
27	0.63	0.69	0.71	0.8
31	0.6	0.85	0.75	0.93
35	0.5	0.64	0.67	0.69

실험 결과, 이 논문에서 제안한 아키텍처 검색 방법과 컴포넌트 검색 방법을 병행하여 검색을 수행할 경우 검색할 컴포넌트의 개수를 줄임으로써 컴포넌트 검색만을

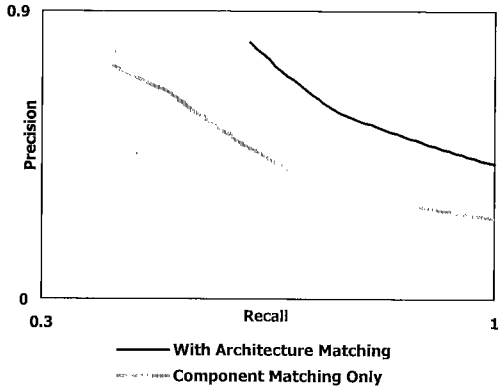

그림 2 제안한 방식과 기존 방식의 비교

수행한 결과에 비해 재현율과 정확성을 높였으며, 아키텍처에 포함된 컴포넌트와 커넥터의 수가 많을수록 정확성이 증가함을 알 수 있다.

시간 복잡성 측면에서, 제안한 검색 방법은 아키텍처 검색과 시그니처 일치 검색 그리고 행위 일치 검색을 병행하므로 높게 평가되었다. 시간 복잡성 측면을 비교 횟수 측면에서 고려하면, 저장된 아키텍처의 총 개수를 A, 아키텍처 내 컴포넌트 수의 평균값을 C_a, 컴포넌트 내 메시지 수의 평균을 M_a, 그리고 저장소에 있는 컴포넌트의 총 개수를 C, 아키텍처 검색 결과 얻어진 후보 아키텍처의 개수를 A_c이라고 하면, 제안된 아키텍처 기반 검색 방식은 다음과 같은 형태의 시간 복잡성을 갖게 되며,

$$O(A \cdot C_a \cdot M_a) + O(A_c \cdot C_a \cdot M_a)$$
$$= O((A + A_c) \cdot C_a \cdot M_c)$$

아키텍처 검색을 제외한 기존 컴포넌트 검색 방식의 시간 복잡성은 다음 식으로 표현할 수 있다.

$O(C \cdot M_a)$ (단, $A >> A_c, C >> C_a$)

제안된 검색 방식은 모든 아키텍처 내에서 컴포넌트와 메시지들을 비교하여 후보 아키텍처를 선정하고, 선정된 아키텍처 내의 컴포넌트와 메시지들을 비교하게 되기 때문에, 모든 컴포넌트의 메시지들을 비교하는 기존 검색 방식과 비교하면 아키텍처의 수와 검색된 후보 아키텍처의 수, 그리고 아키텍처 당 평균 컴포넌트의 수에 따라 시간 복잡성이 결정된다. 시간 복잡성을 고려한 결과 아키텍처를 작게 구성할수록, 그리고 아키텍처 검색의 정확성을 높일수록 검색 효율이 높아짐을 알 수 있다.

5. 결 론

이 논문에서는 컴포넌트 기반 소프트웨어 개발에서 컴포넌트의 효율적인 검색을 위한 컴포넌트 검색 방법을 제안하여 아키텍처에 기반한 컴포넌트 검색 시스템을 설계하고, 프로토타입 검색 시스템을 구현함으로써 실험을 통해 제안한 검색 방법의 타당성을 입증하였다. 제안된 아키텍처 검색 방법은 아키텍처의 기능적인 측면에서 보다는 구조적인 측면에서 개발자의 요구에 일치하는 아키텍처를 검색할 수 있다. 검색된 아키텍처 재구성을 위해 아키텍처 내에 컴포넌트를 대체할 수 있는 컴포넌트를 검색하기 위해서 시그니처 일치 검색 방법과 행위 일치 검색 방법을 제안하였다. 아키텍처 검색을 이용하여 아키텍처를 검색한 후, 아키텍처 재구성을 위해 컴포넌트를 검색하도록 함으로써 컴포넌트 검색의 범위를 제한하여 컴포넌트 검색의 재현율과 정확성을 향상시키고, 일치 수행을 위한 비교 횟수를 줄임으로써 검색 수행 시간을 단축시켰다. 또한, 소프트웨어 개발을 위한 아키텍처와 컴포넌트의 재사용성을 향상시켰다.

그러나, 아키텍처에 포함된 컴포넌트의 개수가 많아짐에 따라 재현율은 많은 향상이 있었으나, 검색 시간과 정확성은 기존의 검색 방법보다 월등한 향상을 보이지는 못했다. 이는 재현율과 정확성 그리고 검색 시간의 단축을 위해서 아키텍처를 구성할 때, 기존에 명세되어 있는 아키텍처들을 재사용하는 계층적인 아키텍처를 구성함으로써 극복할 수 있다. 앞으로 보다 완벽한 컴포넌트 검색 시스템으로의 발전을 위하여 컴포넌트와 아키텍처의 분류 등의 보완 작업을 통해 해당되는 질의의 크기를 조절하는 방법에 대한 연구, 컴포넌트 저장소에서의 컴포넌트들간의 통합 과정, 개발자의 요구에 일치하는 컴포넌트가 검색되었을 때 적합성을 검증하기 위한 체계적인 테스트 과정, 그리고 분산 공동 작업 환경에서의 적용에 관한 추가적인 연구가 필요하다.

참 고 문 헌

[1] C. W. Krueger, "Software Reuse," ACM Computing Surveys, Vol. 24, No. 2, 1992

[2] J. Penix, P. Alexander, "Toward Automated Component Adaptation," In Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering, 1997

[3] 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

[4] A. Podgurski, L. Pierce, "Retrieving Reusable Software by Sampling Behavior," ACM Transaction on Software Engineering Methodology, Vol. 2, No. 3, 1993
[5] 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
[6] M. Shaw, "Architectural Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging," IEEE Symposium on Software Reuse, IEEE Press, New York, 1995
[7] D. E. Perry, D. Garlan, "Introduction to the Special Issue on Software Architecture," IEEE Transaction on Software Engineering, vol. 21, No. 4, Apr. 1995
[8] Rapide Design Team, "The Rapide-1 Architectures Reference Manual," Program Analysis and Verification Group, Computer System Laboratory, Stanford University, Oct. 1994
[9] J. E. Robbins, E. J. Whitehead Jr., N. Medvidovic, and R. N. Taylor, "A Software Architecture Design Environment for Chiron-2 Style Architectures," Arcadia Technical Report UCI-95-01, University of California, Irvine, Jane, 1995
[10] A. Mili, R. Mili and R. Mittermeir, "A Survey of Software Components Storage and Retrieval," The Institute for Software Research Technical Report, Dept. of Computer Science and Electrical Engineering, West Virginia University, Oct. 17, 1997
[11] J. Penix, "Automated Component Retrieval and Adaptation Using Formal Specifications," Ph. D Dissertation, Dept. of Electrical and Computer Engineering and Computer Science, University of Cincinnati Apr. 1998

이 승 근
1992년 ~ 1996년 인하대학교 전자계산공학과(이학사). 1996년 ~ 1998년 인하대학교 대학원 전자계산공학과 졸업(이학사)
1998년 ~ 현재 인하대학교 대학원 전자계산공학과 박사과정 수료. 관심 분야는 CSCW, 소프트웨어공학, 이동 에이전트 시스템.

안 치 돈
1990 ~ 1995년 인하대학교 전자계산공학과(공학사). 1995년 ~ 1997년 인하대학교 대학원 전자계산공학과 졸업(공학석사). 1997년 ~ 현재 인하대학교 대학원 전자계산공학과(박사과정). 관심분야는 분산멀티미디어, 멀티미디어 표준화, 소프트웨어공학.

이 윤 수
1988년 인하대학교 전자계산학과 졸업(학사). 1990년 인하대학교 대학원 전자계산학과(이학석사). 1997년 인하대학교 대학원 전자계산공학과(박사과정수료). 1995년 ~ 1996년 인하대학교 전자계산공학과 전임대우강사. 1997년 ~ 현재 안산공과대학 전자계산소장. 1996년 ~ 현재 안산공과대학 전산정보과 조교수. 관심분야는 컴포넌트 기반 소프트웨어 공학, 분산객체 컴퓨팅, 원격교육.

왕 창 종
1958년 ~ 1964년 고려대학교 물리학과(이학사). 1973년 ~ 1975년 성균관대학교(경영학석사). 1979년 ~ 현재 인하대학교 전자계산공학과 교수. 관심분야는 소프트웨어공학, 분산객체기술, 컴퓨터기반교육.