자동학습에 기반한 디자인 패턴 인식

황성욱[1], 윤현상[2], 이은석[3]
성균관대학교 컴퓨터공학과[1 2 3]
{hwangsw[1], wizehack[2],eslee[3]}@selab.skku.ac.kr

Design Pattern Discovery based on Machine Learning

Sungwook Hwang[1], Hyunsang Youn[2], Eunseok Lee[3]
Dept. of Computer Engineering, Sungkyunkwan University[1 2 3]

요 약

디자인 패턴의 사용은 시스템을 좀 더 유연하고, 이해하기 쉽고, 재사용 가능하게 만든다. 개발이 완료된 시스템이 명확하게 문서화가 되어 있으면, 시스템의 내부 구조를 이해하기 쉽고, 향후 유지 보수의 비용이 적게 든다. 하지만, 대부분 시스템의 경우 개발된 시스템의 문서화가 잘 되어 있지 않기 때문에, 시스템에 문제가 생겨 수정 하고자 하거나, 새로운 모듈을 추가하여 시스템의 기능을 확장하고자 할 때, 전체 소스코드를 분석하여 시스템을 이해해야 하는 어려움이 있다. 이러한 문제점을 해결하기 위해서 소스코드에서 자동적으로 디자인 패턴을 인식하여 문서화를 증진시킬 수 있다. 따라서 신뢰할 만한 디자인 패턴 인식 시스템이 중요하다. 지금까지 디자인 패턴 인식 방법은 시스템의 구조적인 특징만을 이용하여 패턴을 인식하여 왔다. 그래서 본 논문은 구조적인 특징뿐만 아니라 동적인 분석, 그리고 자동학습(machine learning)에 기반하여 소스코드로부터 디자인 패턴을 인식하는 방법을 제안하고자 한다. 그리고 전 작업에서 만든 자바로 쓰여진 에이전트 개발 툴을 대상으로 실시하여 인식한 디자인 패턴에 대한 평가를 하였다.

Keyword : 디자인패턴 인식, TVL, 정적분석, 동적분석, 자동학습

1. 서 론

오늘날 디자인 패턴의 주요 목적은 이용 할 수 있는 소프트 아키텍처의 디자인의 공통된 측면을 추출하고, 어떤 특정 디자인 문제에 대한 최적화된 해결책을 만드는 것이다.[1]

디자인 패턴의 사용은 시스템을 좀 더 유연하고, 이해하기 쉽고, 재사용 가능하게 만든다. 패턴을 통해 코드를 쉽게 이해할 수 있고, 전체 코드를 수정하지 않고 부분적으로 코드를 쉽게 수정할 수 있다. 이러한 개발이 완료된 시스템에 관하여 문서화가 되어 있으면, 소스코드를 이해하기 쉽고, 향후 유지 보수의 비용이 적게 든다. 하지만, 대부분 시스템의 경우 개발된 시스템의 문서화가 잘 되어 있지 않기 때문에, 시스템에 문제가 생겨 수정 하고자 하거나, 새로운 모듈을 추가하여 시스템의 기능을 확장하고자 할 때, 전체 소스코드를 분석하여 시스템을 이해해야 하는 어려움이 있다. 이러한 문제점을 해결하기 위해서 소스코드에서 자동적으로 디자인 패턴을 인식하여 문서화를 증진시킬 수 있다. 따라서 신뢰할 만한 디자인 패턴 인식 시스템이 중요하다.

지금까지 디자인 패턴 인식 방법은 디자인 패턴의 구조적인 특징만을 이용하여 패턴을 인식하여 왔다. 그러나, 가령 GoF 패턴 중 State 패턴과 Strategy 패턴과 같이 서로 유사한 구조를 가진 패턴의 경우 서로 인식하기가 힘들고, 또한 하나의 디자인 패턴에 대하여 구현을 할 때 다양한 소스코드의 형태로 나타나기 때문에 그 패턴을 인식하는 것도 쉽지가 않다. 그래서 본 논문은 구조적인 특징뿐만 아니라, 동적인 분석, 그리고 자동학습(machine learning)에 기반하여 소스코드로부터 디

자인 패턴을 인식하는 방법을 제안하고자 한다.

패턴을 인식하는 대략적인 과정을 보면, 먼저 디자인 패턴의 구조적인 측면을 TVL(Tag Value Language)과 클래스 다이어그램을 이용하여 형식적으로 기술한 후 저장한다. TVL 은 정적인 분석을 할 때 유용하고, 또한 정적/동적인 분석을 통해 얻어진 패턴 후보를 대상으로 학습을 하고자 할 때 유용한 factor 로 사용된다. 그 후, 정적인 분석을 통해 소스코드로부터 얻어진 ASG(Abstract Syntax Graph)와 저장된 패턴 구조를 매치하여 정적인 패턴 후보들을 생성한다.

그러나 정적인 분석만으로는 다형성이나 동적 메소드 바인딩과 같은 실질적인 메스드 호출과 오브젝트는 인식하기가 어렵다. 그래서 정적인 분석을 통해 생성된 패턴 후보들을 대상으로 동적인 분석을 실행한다. 동적인 분석을 통해 생성된 패턴 후보들을 대상으로 자동학습의 ID3 을 이용하면 패턴을 좀 더 세밀하게 인식할 수 있다. 이렇게 ID3 알고리즘을 이용하여 훨씬 효율적인 디자인 패턴 인식을 할 수가 있다. 본 논문에서는 전 작업에서 만든 자바로 쓰여진 에이전트 개발 툴[8]을 대상으로 실시하여 인식한 디자인 패턴에 대한 평가를 하였다.

본 논문은 다음과 같이 구성되었다. 다음 장에서 우리는 우리 연구의 목적과 유사한 특징을 가지고 있는 다른 연구에 대한 개요를 살펴보고, 각각의 장/단점을 파악한다. 3 장에서는 정적인 분석, 동적인 분석, 학습을 포함하여 우리가 제안하는 툴에 관하여 살펴 보고, 4 장에서는 3 장에서 제안한 툴을 이용하여 우리 연구의 결과를 보여줄 것이다. 마지막으로 결론을 내리고, 미래 연구에 대한 제안을 한다.

2. 관련연구

소프트웨어 시스템의 구현으로부터 디자인 패턴을 인식하는 것과 관련한 다른 접근을 알아보자.

FUJABA[2]는 디자인 패턴이 UML 클래스 다이어그램과 스토리 다이어그램을 통해 정의되고, 디자인 패턴의 공통의 부분이 sub-patterns 으로써 분리되어 정의된다. 다른 sub-patterns 으로부터 패턴의 생성은 상속과 use 관계를 이용한다. FUJABA 에서 sub-patterns 의 역할은 디자인 패턴 카탈로그의 범위를 줄이고, 인식 과정에서 atomic 요소로 재사용되고, 소스코드와 디자인 패턴 사이에서 추상적인 레벨을 만든다. FUJABA 의 장점은 패턴과 서브패턴이 UML 다이어그램을 통해 정의되기 때문에 소프트웨어 엔지니어가 이해하고, 사용하기가 쉽고, 서브 패턴을 정의하기 때문에, 패턴 자체를 인식하는 것보다 덜 복잡하다. 단점으로는 어떻게 서브 패턴이 식별되어지는가에 대한 기술이 부족하고, 서브패턴의 카탈로그가 제공되어 있지 않고, 인간이 개입하여 툴을 통제하는 반자동적인 형태를 가지고 있다.

SPQR[3]에 대하여 살펴보면, 디자인 패턴은 두 가지 요소를 통해 정의되는데, EDPs(Elemental Design Patterns), Rho-calculus 이다. EDPs 는 근본적인 OO 개념을 형식적으로 표시한 의미로 표현한 낮은 레벨의 디자인 패턴이고, 디자인 패턴은 EDPs 의 복합으로 표현된다. 그리고 Rho-calculus 는 sigma calculus 의 부분집합과 reliance operator 로 구성된다. Rho-calculus 는 sigma calculus 로부터 타입 정의, 오브젝트 타이핑, 타입 subsumption 개념을 상속하고, reliance operator 는 요소가 다른 요소를 의존하고 있는지를 표현한다. EDPs 의 역할은 SPQR 의 중심적인 요소이고, 디자인 패턴으로서 도메인과 언어에 무관하고, EDPs 를 인식하는 것은 디자인 패턴을 인식하지 않고 소스코드를 좀 더 이해할 수 있도록 한다. SPQR 의 장점은 Rho-calculus 를 이용하여 EDPs 를 기술하기 때문에 formal 한 메커니즘을 가지고, EDPs 를 통해 기술되어진 디자인 패턴은 덜 복잡하고, 이해하기가 쉽다. 한편 단점으로는 구조적인 관계를 추출하는 것은 수월하지만, 관계의 구현에서 variation 에 의해 복잡해지고, EDPs 들이 어떻게 디자인 패턴으로 구성되어지는지에 대한 기술이 없다.

한편, [4]는 디자인 패턴 인식 툴을 인식과정 동안 사용하는 정보에 기반한 인식 툴을 분류했다.

그림 1. 디자인 패턴 인식 과정

- 디자인 패턴을 전체로 고려하는 툴(PTIDEJ [5], CrocoPat [6]); 이것은 매칭 알고리즘이 소스 코드와 전체 패턴을 매핑하고자 하는 것을 뜻한다.
- 디자인 패턴이 구성하는 최소한의 키 구조를 고려하는 툴(SPOOL [7]); 이것은 매칭 알고리즘이 패턴인 만들어지는 구조의 핵심을 구별하는 것을 뜻한다.
- 디자인 패턴이 만들어지는 서브 컴포넌트를 고려한 툴(FUJABA, SPQR); 이것은 패턴의 서브 컴포넌트를 구별하는 것에 의하여 점진적으로 작업한다는 것을 뜻한다.

이상 3 부분으로 고려를 했으나, 전부 구조적인 측면만을 분석했다. 따라서 우리는 다음 장에서 구조적인 측면 뿐만 아니라 동적인 측면까지 고려하여 시스템을 제안하고자 한다.

3. 제안 시스템

앞 장에서 언급한 관련 논문들이 패턴의 구조적인 측면만을 분석하여 패턴을 인식했으므로 우리는 구조적인 측면 뿐만 아니라 동적인 측면, 학습까지 고려한다. 이 장에서는 그러한 모든 측면들을 고려하여 어떻게 소스코드로부터 디자인 패턴을 인식하는지에 대한 전체적인 디자인 패턴 인식 과정 소개하고자 한다.

그림 1.과 같이 패턴 인식과정의 단계는 먼저 소스코드로부터 필요한 정보를 추출한다. 추출된 정보는 ASG(Abstract Syntax Graph)와 같이 언어와 무관한 방법으로 표현된다. 다음은 정적인 분석을 수행하는데, input 으로써 ASG 이외에 TVL 과 클래스 다이어그램을 이용하여 형식적으로 기술된 디자인 패턴 카탈로그가 필요하다. 인식 알고리즘은 소스코드로부터 추출된 정보와 카탈로그에 저장된 디자인 패턴을 매핑하고자 하는 매칭 기술에 기반하고 있다. 정적인 분석을 통해 생성된 패턴후보는 동적인 분석을 위한 input 으로 들어간다. 동적인 분석은 정적인 분석만으로 분석할 수 없는 동적인 행위를 조사하여 좀 더 정확한 결과를 만든다. 동적인 분석을 통해 생성된 결과는 학습을 위한 대상이 된다. 학습을 통하여 더 정확한 디자인 패턴 인식의 결과를 얻을 수 있다.

3.1 정적인 분석

소스코드는 정적인 분석을 하기 위해서는 기본이다. 공통의 컴파일러에 의해 계산되어진 ASG 의 형태로 나타난다. 그리고 인식하고자 하는 패턴은 TVL(Tag Value Language)과 함께 클래스 다이어그램으로 기술된다.

TVL[9]은 Perl 언어의 간단한 부분집합이다. 이것을 이용하여 언어나 도메인에 무관하게 디자인 패턴을 형식적으로 기술할 수 있다. 그리고 Perl 언어는 널리 사용된 언어이기 때문에 많은 사용자들에게 친숙하다. 또한 Perl 언어는 이용 가능한 프리웨어와 shareware 에 의해 지원되는 장점이 있다.

TVL 은 number 와 같이 간단한 literal 을 표현할 수 있고, 변수와 대수적인 함수를 포함한 복잡한 표현을 할 수도 있다. 하지만 표현이 무엇이든지, 평가되어질 때, 기대된 값이 만들어진다.

Stereotype	Base Class	Tags
<<PDsingleton>>	Child Class	PDInctance

Tags	Type	Multiplicity	D.A.Name
PDInctance	Boolean	[0..n]	Singleton:: getInstance

표 1. stereotype 과 tag 기술

표 1 에서와 같이 TVL에 사용되는 stereotype과

tag를 표시했다. 클래스다이어그램의 주석에 들어가는 스테레오타입과 기본 클래스, tag의 명칭과 tag의 타입, tag의 다중성, 그리고 tag의 Domain Attribute Name을 기술했다. 이것은 TVL로 기술되어 패턴들간의 분류를 할 수 있게 한다. 우리는 <<PDsingleton>> 스테레오타입 이외에도 패턴 인식을 위한 전체 스테레오타입을 기술했다.[1]

가령, 아래 그림 2, 그림 3 과 같이 State 패턴과 Strategy 패턴은 구조적으로 구분할 수 없다.

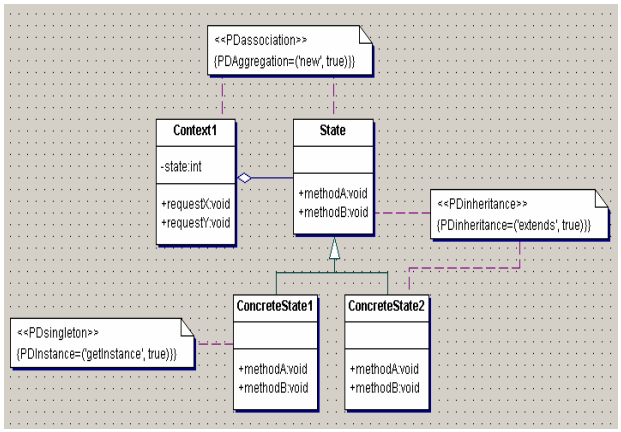

그림 2. State 패턴 클래스 다이어그램

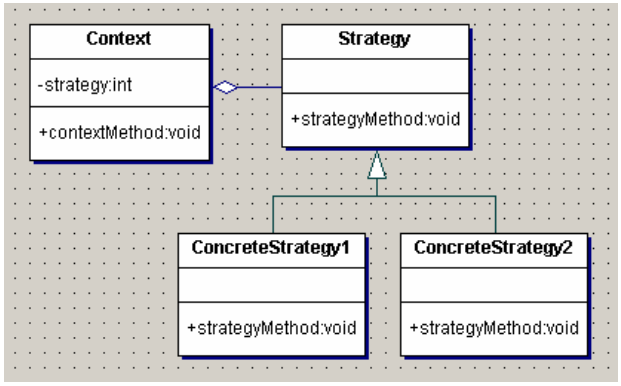

그림 3. Strategy 패턴 클래스 다이어그램

하지만 많은 실제 구현에서 State 패턴은 Singleton 패턴과 같이 혼용해서 사용하는 경우가 많다. 그래서 그림 2 와 같이 TVL 을 이용하여 getInstance( )라는 메소드를 찾을 수 있도록 기술하여 많은 false positive 를 줄일 수 있다. 이러한 TVL 은 나중에 설명하게 될 패턴인식을 위한 마지막 단계인 학습에도 유용하게 사용되는 factor 이다. 학습을 통해서 더 많은 false positive 를 줄이고 패턴 인식의 효율을 더 높인다.

TVL 이외에도 정적인 분석을 위해서는 디자인 패턴의 기본적인 요소인 Inheritance, Composition, Aggregation, Association 을 고려한다. Inheritance 는 한 클래스가 다른 클래스로부터 비롯되는 것을 뜻하고, Composition 은 한 클래스가 데이터 변수에 의해 직접적으로 다른 클래스 인스턴스를 포함하고 있는 것을 뜻하고, Aggregation 은 한 클래스가 참조나 포인터를 가지는 것에 의해 다른 클래스를 간접적으로 포함하고, Association 은 한 클래스가 오퍼레이션 파라미터나 혹은 리턴 타입으로 반환되는 다른 클래스를 사용하고 있다는 것을 뜻한다.

이렇게 기술된 패턴은 표준 XML DOM tree[10]로 로드되어 ASG 와 매치되어 정적인 패턴 후보들을 생성한다.

3.2 동적인 분석

GoF[1] 디자인 패턴은 가령, 동기, 적용, 결과, 구현과 같이 대부분이 비형식적이다. 패턴 기술의 형식적인 부분은 구조와 때로는 협력 방법이다. 협력방법은 때로는 패턴의 구성요소들의 행위를 나타내는 UML 시퀀스 다이어그램을 포함한다.

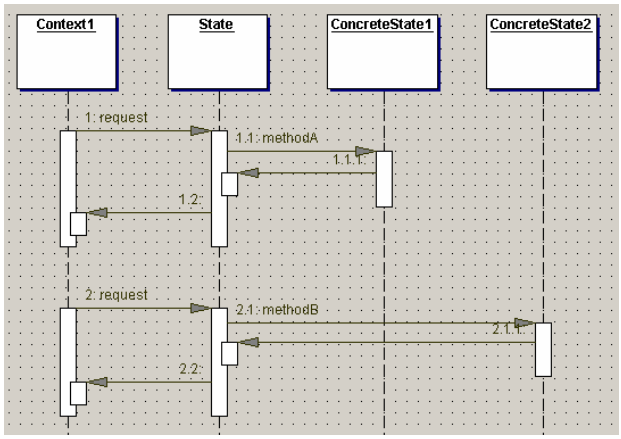

그림 4. State 패턴 시퀀스 다이어그램

앞에서 수행한 정적인 분석은 다형성이나 동적 메소드 바인딩과 같은 실질적인 메스드 호출과 오브젝트를 인식하기가 어렵다. 따라서 UML 시퀀스 다이어그램에 기반하여 객체들의 동적인 행위를 분석하고자 한다. 이에 따라 앞에서 수행한 정적

[1] 전체 스테레오타입은 http://selab.skku.ac.kr/PatternDiscovery에 기술되어 있다.

인 분석의 결과인 패턴 후보들을 대상으로 동적인 분석을 한다.

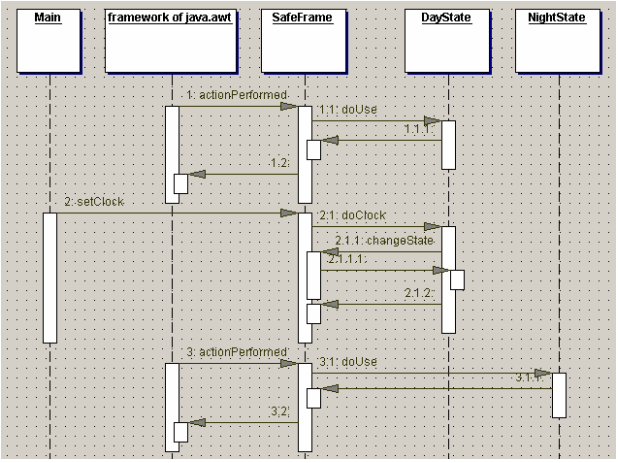

그림 5. State 패턴의 Method trace

분석과정은 프로그램 실행 동안 디버깅을 하여 객체 정보와 메소드 trace 를 기록하여 call graph 로 저장되어 패턴 후보 인식을 위한 데이터를 구성한다. 그리고 패턴에 대한 시퀀스 다이어그램과 매칭을 하여 동적인 패턴 후보가 생성된다.

3.3 학습

우리는 정적, 동적인 분석의 결과로부터 습득된 패턴 후보들에 대하여 자동학습 시스템을 이용했다. 자동학습은 그 분야에서 가장 알려진 ID3 알고리즘을 이용했다. ID3 의 생성 알고리즘을 보면,

- 단계 1 : 한 attribute 를 tree 의 루트 노드로 선정하고 이 attribute 가 가질 수 있는 모든 서로 다른 값을 가지고 가지를 만든다.
- 단계 2 : 이 tree 는 트레이닝 set 을 분류하는데 사용된다. 어느 특정 리프에서 모든 예가 같은 클래스에 속한다면 이 리프는 이 클래스로서 레이블 된다. 모든 리프가 클래스로서 레이블 되면 알고리즘은 끝난다.
- 단계 3 : 그렇지 않으면 이 노드는 루트까지의 패스상에 나타나지 않는 attribute 로서 레이블 되고 모든 가능한 값들에 대해 가지가 생성된다. 이 알고리즘은 2 단계로 가서 계속된다.

이렇게 정적인 분석에서 기술한 TVL 을 기반으로 반복적인 학습을 통해 좀 더 정확한 패턴 인식 결과를 얻는다.

4. 평 가

이장에서는 전 작업에서 만들었던 툴[8]을 기반으로 하여 디자인 패턴 인식 결과를 가지고 우리가 제안하는 툴의 효율성에 대한 평가를 실시한다. 툴은 패턴 기반 에이전트 개발을 지원하는 도구로써 표 2 와 같은 툴의 크기 정보를 나타낸다. 한편, 평가요소는 다음과 같다.

- Recall[11] : 시스템에서 모든 구현된 패턴의 수가 인식된 패턴의 수에 의해 나누어진 값. 가령, 100%의 recall 은 적어도 모든 구현된 패턴이 인식되었다는 것을 뜻함.
- Precision[11] : 인식된 패턴의 수에 의해 나누어진 인식되고 실제로 구현된 패턴의 비. 가령, 50%의 precision 은 인식된 패턴의 절반은 시스템에서 실제적으로 구현되지 않았다는 것을 뜻함.

이 두 가지 요소를 가지고 패턴 인식 툴을 평가했다.

파일 수	크기	라인수
72	3.1MB	92,782

표 2. 툴의 크기 정보

그리고 표 3 에서는 State 패턴에 대하여 인식한 결과를 나타내었다.

평가요소	정적/동적 분석	ID3
recall	47%	78%
precision	53%	82%

표 3. 분석 결과

우리는 정적/동적인 분석과 ID3 알고리즘을 이용한 자동학습의 분석을 수행하여 결과를 측정했다. 표 3 에서와 같이 정적/동적인 분석보다는 정적/동적 분석 후 학습을 이용하는 것이 패턴 인식 툴에 대한 recall 과 precision 의 value 가 훨씬 높다는 것을 알 수 있다.

5. 결 론

디자인 패턴의 사용은 시스템을 좀 더 유연하고, 이해하기 쉽고, 재사용 가능하게 만들고, 또한 시스템의 질을 측정할 수 있다. 디자인 패턴에 관하여 문서화가 되어 있으면, 소스코드를 이해하기 쉽고, 필요에 따라 코드를 보다 빠르게 수정 할 수 있다. 하지만, 대부분 시스템의 경우 디자인 패턴의 문서화가 잘 되어 있지 않기 때문에, 시스템에 문제가 생겨 수정 하고자 하거나, 새로운 모듈을 추가하여 시스템의 기능을 확장하고자 할 때, 전체 소스코드를 분석하여 시스템을 이해해야 하는 어려움이 있다. 이러한 문제점을 해결하기 위해서 소스코드에서 자동적으로 디자인 패턴을 인식하여 문서화를 증진시킬 수 있다. 따라서 신뢰할 만한 디자인 패턴 인식 시스템이 중요하다. 지금까지 디자인 패턴 인식 방법은 디자인 패턴의 구조적인 특징만을 이용하여 패턴을 인식하여 왔다. 그래서 본 논문은 자동학습(machine learning)에 기반하여 소스코드로부터 디자인 패턴을 인식하는 방법을 제안하고자 한다. 그리고 전 작업에서 만든 자바로 쓰여진 에이전트 개발 툴을 대상으로 실시하여 인식한 디자인 패턴에 대한 평가를 하였다.

현재는 우리는 일부 패턴에 대해서만 평가를 실시하였기 때문에 전체 GoF 패턴에 대해서는 평가를 실시하지 못했다. 그래서 향후에는 전체 GoF 패턴에 대한 평가를 실시할 예정이다.

참고문헌

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: elements of reusable object-oriented software, Addison Wesley, Reading MA, USA, 1994.

[2] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh, “Towards Pattern-Based Design Recovery”, Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida, USA, 2002, pp. 338-348.

[3] J. McC. Smith, and D. Stotts, “SPQR: Flexible Automated Design Pattern Extraction From Source Code”, Proceedings of the 2003 IEEE International Conference on Automated Software Engineering, Montreal QC, Canada, October, 2003, pp. 215-224

[4] Francesca Arcelli Fontana, Stefano Masiero, Claudia Raibulet, Francesco Tisato: A Comparison of Reverse Engineering Tools Based on Design Pattern Decomposition. Australian Software Engineering Conference 2005: 262-269

[5] H. Albin-Amiot, P. Cointe, Y. G. Guéhéneuc, and N. Jussien, “Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together”, Proceedings of the 16th International Conference on Automated Software Engineering, San Diego, CA, USA, 2001, pp. 166-173.

[6] D. Beyer, and C. Lewerentz, “CrocoPat: Efficient Pattern Analysis in Object-Oriented Programs”, Proceedings of the 11th IEEE International Workshop on Program Comprehension, Los Alamitos, CA, USA, 2003, pp. 294-295.

[7] R. K. Keller, R. Schauer, S. Robitaille, and P. Page, “Pattern-Based Reverse-Engineering of Design Components”, Proceedings of the International Conference on Software Engineering, Los Angeles, CA, USA, 1999, pp.226-235.

[8] Hyunsang Youn, Sungwook Hwang, Hee Yong Youn, Eunseok Lee: Next Generation Agent Development Supporting Tool: Case Study. SERA 2005: 264-273

[9] http:/www.omg.org.

[10] World Wide Web Consortium (W3C). Document Object Model (DOM), 2000.

[11] I. Philippow, D. Streitferdt, M. Riebisch, and S. Naumann, “An Approach for Reverse Engineering of Design Pattern”, Software and Systems Modeling, Springer Verlag, April 2004.