소스코드 식별자의 일관적 명명을 위한 데이터 마이닝 적용 방안

강원대학교 ▌김 순 태*
소프트웨어 엑스퍼트 그룹 ▌심 빈 구

1. 서 론

소프트웨어 개발자나 소프트웨어 유지 보수 담당자는 소프트웨어를 이해하기 위하여 소스 코드 식별자(Identifier)와 코드 내 주석(Comment)을 의존한다. 하지만 코드 내의 주석이 충실히 되어 있는 경우가 많지 않아 결국 유지보수 담당자는 코드의 식별자를 기반으로 프로그램을 이해한다[1]. 식별자는 개발자가 명명하는 클래스, 메소드, 필드, 파라미터 등의 코드 구성 요소의 고유한 이름으로 해당 구성 요소의 특성과 행위 등을 표현한다. 그래서 적절히 명명하지 않거나, 비 일관적으로 명명한 식별자는 소프트웨어의 가독성을 떨어트리게 된다[2]. 게다가, 다수의 개발자가 장시간에 걸쳐 개발하는 소프트웨어에서 비 일관적으로 명명된 식별자는 흔하게 발견된다. 이는 개발자 마다 표현 방식이 다르고, 시간에 따라서 요구사항 등의 구현이 달라지기 때문이다.

비 일관적인 식별자를 검출하기 위하여 여러 연구가 진행되어 왔다([2-6]). 이는 크게 두 가지 종류로 분류할 수 있다. 첫째, 식별자와 식별자가 의도하는 개념 사이의 매핑 정보를 기반으로 비 일관적 식별자를 검출하는 방법으로[2-4], 식별자의 구성 패턴과 WordNet[7]을 활용하여 동음 이의어(Homonym)와 동의어 (Synonym)를 식별하였다. 둘째, 식별자로부터 온톨로지 정보를 구축하여 코드에 대한 이해를 높이고, 이를 추후에 식별자 명명에 사용하기 위한 접근 방법이 있다[5-6]. 이 방법은 현재 코드의 비 일관 식별자 검출 보다는 분석 이후 비 일관 식별자를 줄이는데 기여할 수 있다. 본 연구는 첫 번째 접근 방법의 확장으로 생각될 수 있다.

본 논문에서는 자연어 처리 기법(NLP: Natural Language Processing)을 사용하여 Java 소스 코드에서의 비 일관적인 식별자를 검출하는 기법을 소개한다. 이 기법은 1) 모든 코드 구성 요소를 Java의 명명 규칙을 기반으로 형태소를 분석하여 식별자를 구성하는 용어(Term)를 추출하고, 구문 분석기[8]를 통하여 각 용어의 품사(POS: Part of Speech)를 분석한다. 2) 각 용어는 온톨로지를 사용하여 의미상 유사어(Semantic Similar Word; Synonym)를 식별하고, 용어 내 문자간 거리 측정 알고리즘[9]을 사용하여 구조상 유사어(Syntactic Similar Word)를 추출한다. 3) 마지막으로, 제안된 규칙을 통하여 용어의 품사, 의미 그리고 구조상의 비 일관 식별자를 검출한다.

본 논문의 기여는 다음과 같이 요약할 수 있다. 첫째, 코드에 대한 배경지식이 없는 개발자도 비 일관성 식별자 검출을 용이하게 할 수 있다. 둘째, 코드의 전수 검사를 통하여 비 일관성 식별자 검토의 누락 비율을 감소시켜 코드의 유지 보수성을 높일 수 있다. 셋째, Eclipse기반 Plugin인 CodeAmigo를 개발하여 본 논문의 제안 기법을 지원하기 위한 도구를 개발하였다. 그리고 이 지원도구를 Java 기반 오픈 소스 프로젝트에 적용하여 검출된 비 일관 식별자를 예시하였다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 Java 명명 규칙과 비 일관 식별자에 대한 배경 지식을 소개한다. 3장에서는 자연어 처리 기법을 사용한 Java 소스 코드 내의 비 일관 식별자 추출 기법을 제안하고, 4장에서는 지원 도구인 CodeAmigo와 오픈 소스 프로젝트에 적용한 사례를 소개한다. 5장에서는 소스코드의 비 일관 식별자를 검출하기 위한 관련 연구를 소개하고, 마지막으로 6장에서는 결론 및 향후 연구에 대하여 논의한다.

2. Java 명명 규칙과 비 일관 식별자

2.1 Java 명명 규칙

Java Code Convention[10]에서는 Java소스 코드 식별자의 명명 규칙을 다음과 같이 정의하고 있다.

* 정회원

- **클래스와 인터페이스:** 구성 용어는 명사(구)를 이루어야 하며, 첫 글자는 대문자여야 한다.
- **메소드:** 구성 용어는 동사(구)를 이루어야 하며, 첫 글자는 소문자 여야 한다.
- **변수:** 명사(구)를 이루어야 하며, 첫 글자는 축약어(e.g., URL, HTTP)이외에 소문자여야 한다.
- **상수:** 모두 대문자이다.

이와 함께 두 개 이상의 용어로 이루어진 코드 식별자 명명 시 Camel Case(대소문자 혼합)를 사용하고, 상수의 경우 모든 용어를 대문자로 쓰고 용어 간의 구분은 underscore로 구분한다. 예를 들어, Class 이름 WhitespaceTokenizer의 경우 Whitespace와 Tokenizer로 이루어진 용어의 집합으로 명사구를 이루며, Method이름 getElementNameForView()는 get, Element, Name, For, View 네 용어로 동사구를 이룬다. 또한 상수의 경우 'MIN_COUNT'의 경우 'MIN'과 'COUNT'의 단어로 이루어져 있다.

위와 같이 Camel Case와 underscore를 사용하여 명시적으로 분리되는 식별자를 hard word라 하고, 분리되지 않는 경우(예, whitespacetokenizer)는 soft word라 한다[3]. Java이외의 다른 언어의 경우 이에 대한 제약사항이 상대적으로 약하기 때문에 Soft word로부터 용어를 식별하는 것은 연구 이슈 중 하나이다[1]. 하지만, Java언어의 경우 Camel Case와 underscore를 사용한 hard word를 명명 규칙으로 정하고 있고, 타 언어에 비하여 잘 준수하고 있다.

2.2 비 일관 식별자 정의

비 일관 식별자란 클래스, 메소드 등의 코드 구성 요소의 식별자를 의미 혹은 구조상 일관적이지 않게 명명하여 코드의 가독성을 떨어트리는 문제를 의미한다[2]. 식별자(Identifier)는 하나 혹은 둘 이상의 용어(Term)로 이루어진 복합어(Compound)이며, 용어가 일반적인 영어 단어 사전(Dictionary)에 존재하는 경우에 이를 단어(Word)라 한다[4]. 코드 구성 요소의 식별자가 비 일관적으로 구성되었는지의 파악은 식별자를 구성하는 각 용어가 의미상, 구조상, 그리고 품사의 면에서 일관적으로 사용되는지 분석함으로써 검출할 수 있다.

첫째, 용어의 의미상 비 일관성은 여러 식별자에 동일한 의미를 가진 서로 다른 단어를 사용한 경우를 의미한다. 예를 들면, 메소드 이름 getUserID()와 retrieveUserID()에서 get과 retrieve는 동사로서의 의미는 유사하나 다른 용어이기 때문에 의미상 비 일관성이라고 판단할 수 있다. 이는 여러 사람이 참여하는 소프트웨어 개발에서 흔히 나타나는 현상이며, 프로그램의 이해에 혼란을 야기시킬 수 있다.

둘째, 용어의 구조상 비 일관성은 용어를 구성하는 알파벳이 비슷한 것을 의미한다. 이는 소스 코드에는 축약어가 많이 사용되기 때문에 발생하는 문제로, 예를 들면 argument를 의미하는 args, arg0, arg1 혹은 parameter라는 용어 대신 사용하는 param, param0 등은 용어의 구조상 비 일관성에 속한다. 이는 사전에는 나타나지 않지만, 코드에서는 흔하게 나타나는 용어의 구조상 비 일관성의 예이다. 또한 이는 오타에 의해서 발생할 수도 있다. 오타는 프로그램의 수행과 연관은 없으나, 개발자가 식별자를 이해하는 데에는 방해가 된다.

셋째, 품사의 비 일관성은 단어의 품사가 비 일관적으로 사용된 것을 의미한다. 몇몇 연구자들은 개발자들이 하나의 단어에 대하여 일관된 품사를 사용하는 것을 관찰하였다[11-12]. 예를 들어 free는 동사, 형용사 그리고 부사의 의미를 가지나, 프로그램에서는 동사만의 의미만 사용되었다. 뿐만 아니라, Deißenbock과 Pizka[2], Lawrie et al.[3] 연구에서 하나의 용어가 하나 이상의 개념과 대응된다면 이를 비 일관성으로 정의하였다. 이는 하나 이상의 품사로 사용되는 자연어의 단어일 경우에는 보편적으로 적용될 수 있다. 그러므로, 단어가 다양한 품사로 사용될 경우 역시 비 일관성으로 정의할 수 있으며, 본 논문에서는 이를 품사 비 일관성으로 정의한다. 이는 용어가 자연어 사전에 존재하는 단어일 경우에 한정된다.

품사 비 일관성은 단어의 품사 비 일관성과, 구문의 품사 비 일관성 두 가지로 분류할 수 있다. 단어의 품사 비 일관성이란 한 단어가 소스 코드에서 여러 가지 품사로 활용되는 경우를 의미한다. 예를 들어, Class 식별자CheckAbort에서 abort는 명사로 사용되었다. 하지만, abort()라는 메소드에서 abort는 동사로 사용되었다. 이는 abort의 품사를 혼용해서 사용한 경우이다. 구문의 품사 비 일관성은 클래스와 메소드가 Java의 명명 규칙에 따라서 명사구 혹은 동사구로 정의하지 않은 경우에 발생한다. 만일 클래스 이름이 Aborted라고 한다면 명명 규칙에 따라 이는 명사 혹은 명사구여야 한다. 하지만 이는 형용사이므로 구문의 품사 비 일관성의 경우에 해당된다.

3. 비 일관 식별자 검출 기법

코드내의 비 일관 식별자 검출 기법은 그림 1과 같이 3단계의 절차를 따른다. 1) 모든 코드의 식별자를

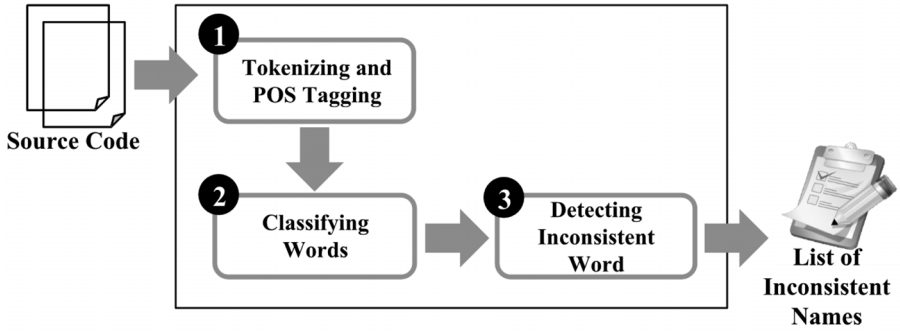

그림 1 비 일관 식별자 검색 절차

명명 규칙을 기반으로 용어로 분리하고, 자연어 구문 분석기를 사용하여 품사를 식별한다. 2) WordNet을 사용하여 각 용어의 의미상 유사어를 분류하고, Levenshtein Distance 알고리즘[9]을 사용하여 구조상 유사어를 분류한다. 3) 비 일관성 용어 검출 규칙을 통하여 비 일관성 용어와 이를 포함하는 식별자를 검출한다.

3.1 식별자의 형태소 및 품사 분석

이 단계에서는 클래스, 변수, 메소드와 파라미터 등의 코드 식별자를 구성하는 용어를 분류한다. 식별자의 형태소 분석은 Java 명명 규칙[10]을 고려하여 수행된다. 식별된 단어는 자연어의 구문 분석기를 사용하여 각 단어의 품사를 분석한다. 코드 식별자는 단일 용어 혹은 하나 이상의 용어로 이루어진 복합어(Compound) 혹은 구(Phrase)를 이루기 때문에 완전하지 않은 문장(Sentence)이다. 그렇기 때문에 구문 분석기를 사용하기 위해서는 구성 요소의 변환이 필요하다.

이를 위하여 먼저 식별자의 모든 용어 간에 공백을 삽입하고, 메소드의 식별자의 경우에만 마지막에 마침표를 추가한다. 이는 메소드는 동사구이므로 명령문과 같이 하나의 문장이 될 수 있기 때문에, 구문 분석의 정확성을 위하여 완전한 문장을 만들도록 변환한다. 이 작업은 특히 명사와 동사의 의미를 함께 가진 하나의 단어로 이루어진 클래스와 메소드 식별자를 처리하기 위하여 중요한 작업이다. 예를 들어 Reset이라는 단어를 클래스 식별자로 단독으로 사용한 경우 개발자는 명사의 의미를 의도하고, 메소드 식별자로 사용한 경우 동사의 의미를 의도하고 사용했을 수 있다.

3.2 의미적 유사어와 구조적 유사어 분류

이 단계에서는 코드 식별자 내의 모든 용어를 의미상 유사어와, 구조상 유사어로 분류한다. 의미상 유사어를 식별하기 위하여 본 연구에서는 WordNet[7]을 활용하였다. WordNet은 영어 어휘 데이터 베이스로, 한 단어의 어근뿐만 아니라, 명사, 동사, 형용사 그리고 부사의 의미와 각 의미 별 동의어 집합을 제공한다.

본 연구에서는 의미상 유사어(Synonym)를 찾기 위하여 전 단계의 용어 식별과 품사 분석의 결과를 활용한다. 이는 하나의 단어는 품사별로 다양한 의미를 가질 수 있고, 동의어는 이 품사별로 존재하기 때문이다. 코드 내의 유사어를 검색하기 위해서는 먼저 용어와 품사 정보를 기반으로 WordNet에서 동의어를 검색하고, 이 동의어가 코드 내에 존재하면 이를 의미상 유사어로 분류한다. 이때, 동의어 또한 동일한 품사여야 한다. 예를 들면, abort가 명사로 쓰인 경우에 WordNet에서는 이의 동의어가 termination, ending, conclusion이 동의어이며, 만약 코드에서 이 단어가 존재하고, 또한 그 단어가 명사로 사용되었다면 의미상 유사 단어로 분류될 수 있다.

구조상 유사어를 식별하기 위하여 Levenshtein Distance 알고리즘[9]을 활용하였다. 이 알고리즘은 두 단어간의 구조적 유사 정도를 측정해주는 알고리즘으로 단어를 구성하는 알파벳의 차를 구하고, 이를 알파벳 수로 나누어 구한다. 예를 들면, kitten과 sitting의 알파벳 차이(D)는 Levenshtein Distance 알고리즘으로 3 (kitten ↔ sitting)이고, 이를 기반으로 단어간의 구조상 유사도는 아래의 수식으로 구할 수 있다. 이때 length는 단어를 구성하는 알파벳의 개수다. kitten과 sitting의 구조 유사 정도는 수식에 따라 0.5(1-3/6)이다.

$$\text{SynSimilarity} = 1 - (D/\max(\text{length}(word1), \text{length}(word2)))$$

3.3 비 일관 식별자 검출 방법

비 일관 식별자 검출 방법의 기본 원리는 다수결 원칙(Majority Rule)이다. 즉, 프로젝트의 참여자 대부분이 사용하는 단어와 품사가 그 프로젝트에서는 일반적으로 동의하는 것으로 판단한다. 다음은 단어, 품사, 유사 단어 분류 결과를 기반으로 식별자의 의미상 비 일관성, 구조상 비 일관성, 품사 비 일관성을 검출하는 방

법에 대하여 소개한다.

3.3.1 의미상 유사어 비 일관 식별자 검출 기법

모든 식별자를 구성하는 용어에 대하여 WordNet을 통하여 동의어를 검색하여 이중에서 의미의 유사 정도가 Threshold 이상(본 논문에서는 0.8로 정함)인 단어를 검출한다. 유사 정도는 WordNet에서 유사어에 대하여 Frequency별로 정렬한 순서를 기반으로 측정한다. 모든 의미상 유사어는 WordNet에서 추출한 유사어들 중에서 소스 코드 내에 존재하는 유사어만을 의미한다. 이를 통하여 계산된 결과는 특정 word와 상당히 유사도가 높은 단어의 집합이다. 이 중에서 가장 코드에서의 출연 빈도가 높은 단어를 기준으로 하여 사용 빈도가 상대적으로 낮은 단어들은 의미의 비 일관성 용어로 검출한다.

예를 들어, 소스 코드에서 동사 get의 동의어 중에서 코드에 존재하는 acquire, grow를 검색했다면, 유사 정도는 get의 동사 의미(sense) 30가지 중 첫 번째 의미인 'come into the possession of something concrete or abstract'과 동의어로 acquire가 존재하고, grow는 열두 번째 의미인 'come to have or undergo a change of physical features'의 의미로써 동의어이다. 그러므로 acquire의 get과의 의미상 유사어 정도(semsim)는 (1-1/(30)) = 0.96이고, grow와의 의미상 유사어 정도(semsim)은 (1-12/30) = 0.6이다. Threshold가 0.8이라고 하면 acquire만이 유사어가 된다. 그 이후 get과 acquire의 사용 빈도를 각각 측정한 후 적게 사용한 단어의 사용처가 의미상의 비 일관성 식별자로 검출 될 수 있다.

3.3.2 구조상 유사어 비 일관 식별자 검출 기법

이 단계의 시작은 용어의 구조상 유사한 단어부터 시작한다. 원리는 구조상 유사한 단어 중 출현 빈도가 적은 것은 구조상 비 일관 식별자로 검출된다. 하지만, 이 접근 방법의 문제점은 코드에서 String accent와 String[] accents와 같이 단수/복수를 구분하여 사용하는 경우가 흔하게 있기 때문에 이 경우도 모두 구조 비 일관성 용어로 검출될 수 있다. 하지만, 이는 코드의 가독성을 높여주는 것이기 때문에 냄새로 판단하기 어렵다. 이 문제를 해결하기 위하여 품사가 명사이고, 해당 단어의 어근이 동일한 경우(accent와 accents의 어근은 모두 accent)는 구조상의 비 일관성 용어 후보 군에서 제외하도록 하였다. 본 논문에서는 용어의 구조상 유사도가 0.8 이상의 단어를 구조상 유사어로 분류한다.

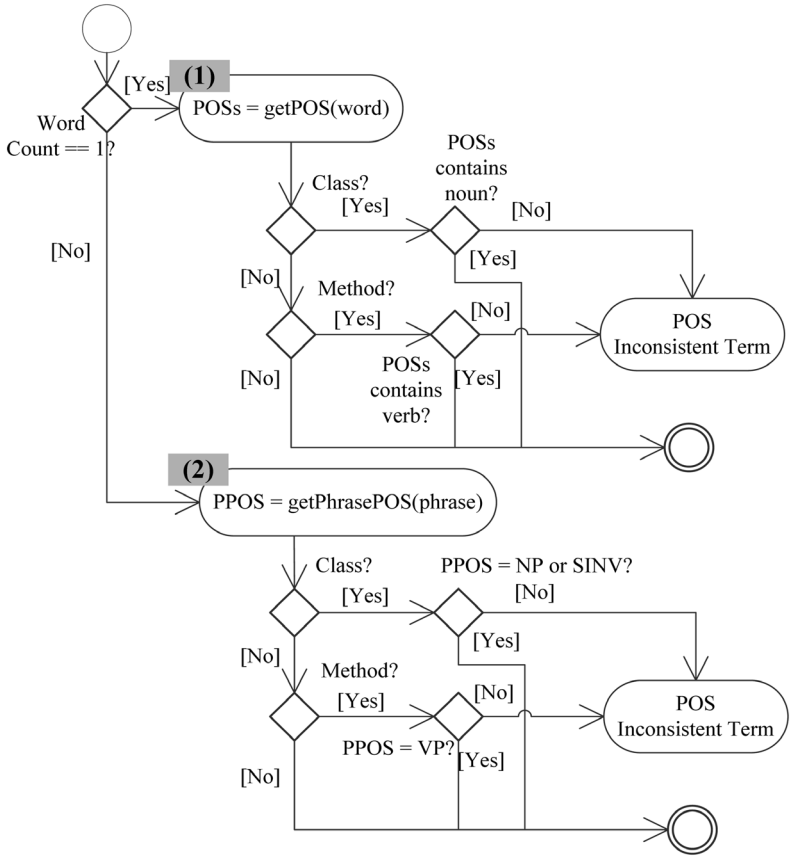

그림 2 구문 품사 비 일관 식별자 검출 알고리즘

3.3.3 품사 비 일관 식별자 검출 기법

품사의 비 일관 식별자 검출 기법은 구문 품사 비 일관성과, 단어 품사 비 일관성 검출의 두 가지 접근 방법으로 검출한다. 첫째, 구문 품사 비 일관성은 구문 분석기의 분석 결과를 기반으로 Java 명명 규칙에서 제시하는 잘못된 품사 사용을 검출한다. 명명 규칙에서는 클래스는 명사 혹은 명사구로, 메소드는 동사 혹은 동사구로 명명하도록 하고 있다. 구문 품사 비 일관성은 클래스 혹은 메소드의 이름을 구성하는 단어가 하나인 경우와 둘인 경우를 분리하여 식별한다. 그림 2는 각 경우에 구문 품사 비 일관성 식별자 검출 알고리즘을 보여준다. 구문 품사 비 일관성은 Threshold가 존재하지 않고 명명규칙이 위배된 경우는 모두 일관성이 위배된 것으로 간주한다.

하나 이상의 용어로 이루어진 복합어는 구를 이룬다. 구로 이루어진 클래스의 식별자인 경우에는 명사구(NP:Noun Phrase)인 경우뿐만 아니라 도치구문(SINV: Inverted decrarative sentence) 역시 타당한 것으로 간주한다. 도치 구문은 동사(구) + 명사(구)의 형식을 의미한다. 이는 클래스 이름의 경우 가장 후반부의 단어가 명사(구) 인 것이 가장 중요하기 때문이다. 예를 들면, 클래스 이름 ReadOnlyDirectoryReader의 경우 Read Only는 동사구를 이루고 Directory Reader는 명사구를 이룬다. 이때 Read Only는 컴퓨터 영역에는 고유명사로 활용되나, 자연어 구문 분석기는 동사구로 인식하기 때문에 이런 경우를 처리하기 위하여 도치 구문 역시 타당한 이름으로 간주한다.

둘째, 단어 품사 비 일관성을 검출하기 위해서 한 단어의 품사별 사용 빈도를 측정 비교하여 Threshold(본 논문에서는 0.8로 정함) 이상 차이가 난다면 사용 빈도가 상대적으로 적은 품사의 사용처가 품사 비 일관 식별자로 검출한다. 예를 들면 abort의 총 출연 횟수가 50회인데, 이중 동사로 사용된 경우가 전체의 80%를 상회할 경우, 다른 품사로 사용된 부분은 품사의 비 일관적 이름으로 판단 될 수 있다.

4. 지원 도구(CodeAmigo) 소개 및 적용

비 일관 식별자를 검출하기 위하여 본 연구에서는 지원도구인 CodeAmigo를 개발하였다. CodeAmigo는 그림 3과 같은 아키텍처로 이루어 졌다. 이는 Eclipse를 기반으로 한 Plugin으로 개발되었으며, 구문 분석과 품사 분석을 위하여 StanfordParser[13]를 활용하였고, 의미상 유사어를 찾기 위하여 WordNet과 지원 라이브러리인 JAWS(Java API for WordNet Searching)[14]와

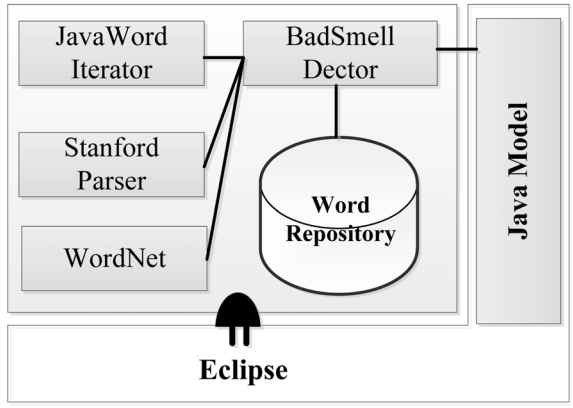

그림 3 CodeAmigo의 소프트웨어 아키텍처

JWI(Java WordNet Interface)[15]를 활용하였다. 또한, 식별된 단어와 구문 분석의 결과 등을 저장하기 위하여 HyperSQL[16]을 활용하였다. Java 소스 코드 구성 요소에 대한 정보를 활용하기 위하여 Eclipse내의 Java Model을 사용하였다.

그림 4는 CodeAmigo의 실행화면을 보여준다. CodeAmigo는 선택한 Eclipse Project내의 모든 소스 식별자를 분석하여 비 일관적 식별자를 식별하고, 그 결과를 Eclipse View로 보여준다. 그 결과는 코드 구성 요소의 타입, 구성 요소 식별자, 비 일관성 타입과, 비 일관성 검출에 대한 설명에 대하여 보여준다. 비 일관성의 타입은 구문 품사 비 일관성, 단어 품사 비 일관성, 의미상 유사어 비 일관성, 구조상 유사어 비 일관성에 대하여 각각, POS-PHR, POS-WORD, SEM, 그리고 SYN으로 표현하였다. 결과에서 각 라인을 선택하면 해당 코드 블록으로 이동하여 코드의 사용처를 직접 조회할 수 있다. 검색 결과의 타당성 검증을 지원하기 위하여 해당 단어가 적절한지를 기록할 수 있도록 체크박스를 배치하였으며, Excel로 모든 정보를 추출하도록 하였다.

본 연구에서는 CodeAmigo를 Java기반 오픈 소스인 Apache Lucene[17]과 Apache Ant[18]의 식별자를 분석하여 비 일관 식별자를 검출하였다. 다음은 각각의 비 일관 식별자를 종류별로 검출한 사례를 보여준다.

• **구문 품사 비 일관성 검출 사례:** 메소드 식별자 readerIndex() - 동사(구)로 이루어져야 할 메소드 식별자에 명사구로 명명, 이때 CodeAmigo는 아래와 같은 설명을 제공한다.

'readerIndex' is used as NP, and composed of[(NP (NP (NN reader)) (NP (NNP Index)) (. .))]. Methods should be named as a verb phrase.

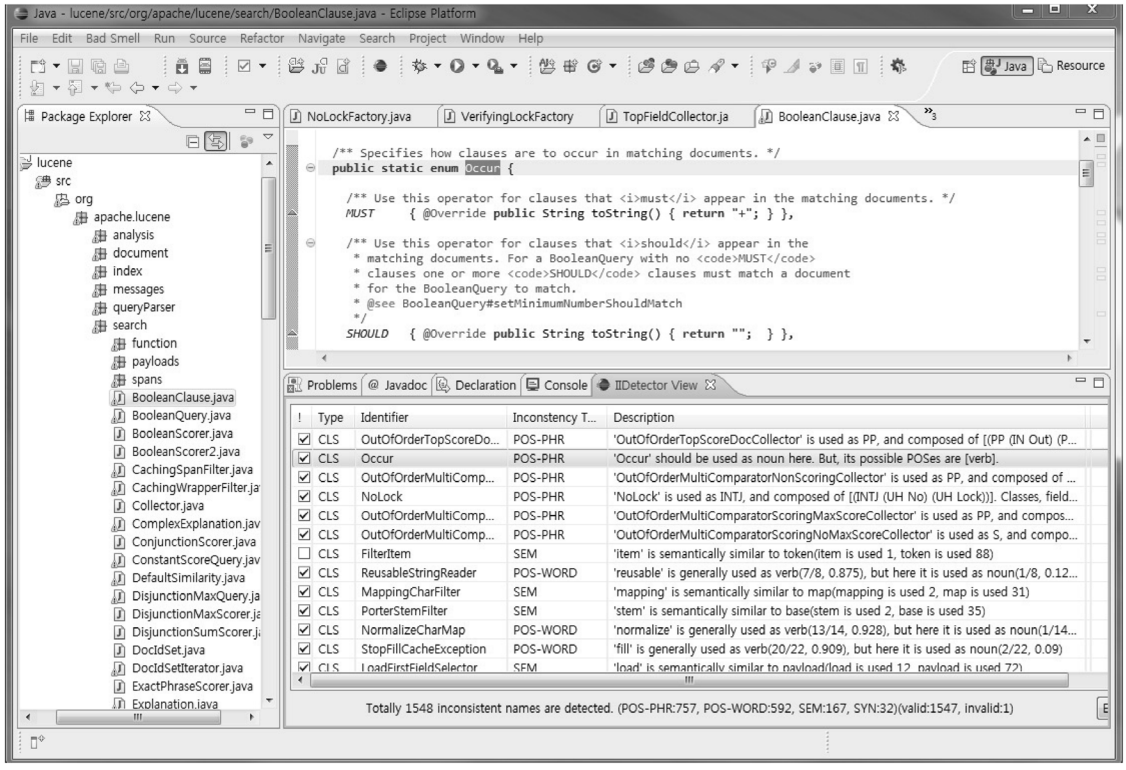

그림 4 CodeAmigo의 화면 캡처

• 단어 품사 비 일관성 검출 사례: 클래스 식별자 CCM-CreateTask - Create는 이 프로젝트에서 동사로 사용되는 경우가 많았는데, 위의 식별자에서는 명사로 사용되었다. 이 식별자에서는 CCMCreationTask가 좀 더 적절한 이름이다. 더불어, CodeAmigo는 create가 동사로 사용된 개수(357개중 341개) 대비, 명사로 사용된 사용 개수(357개중 16개)를 아래의 메시지를 통하여 보여준다.

'create' is generally used as verb(341/357, 0.955), but here it is used as noun(16/357, 0.044).

• 의미상 유사어 검출 사례: Specification과 spec - 축약어와 온전한 단어를 일관적으로 사용하지 않은 사례 역시 검출되었다. 이때 CodeAmigo는 아래와 같은 메시지를 생성한다.

'spec' is semantically similar to specification(spec is used 7 times, specification is used 37 times.)

• 구조상 유사어 검출 사례

메소드 식별자 getPreserve0Permissions() : CodeAmigo는 Perserve0와 Perserve가 구조상 유사하다고 판단하여 다음과 같은 메시지를 생성하였다.

'preserve0' is sytactically similar to preserve(preserve0 is used 1 times, preserve is used 14 times)

5. 관련 연구

Deißenbock와 Pizka[2]는 코드의 식별자의 일관성에 대하여 정형적으로 정의하였다. 이들은 일관성에 대하여 정의하기 위하여 개념과 식별자를 구분하여 하나의 식별자가 두 개 이상의 개념에 대응되는 경우는 동음 이의어(Hononym)로 정의하였으며, 하나의 개념에 두 개 이상의 식별자가 대응되는 경우는 동의어(Synonym)로 정의하였다. 예를 들면, File은 File Name, File Handle의 두 의미를 가질 수 있는 동음 이의어이며, account number와 number는 account number를 의미하는 동의어이다. 또한 지원 도구인 IDD(IDentifier Dictionary)를 소개하여 프로젝트 내에 일관성 규칙에 위배된 식별자에 대한 경고 정보를 보여주었다. 하지만, 이 접근 방법에서 개념과 식별자간의 대응 관계는 개발자가 수동으로 연결시켜주어야 한다는 단점을 가지고 있다.

Lawrie et al.[3]은 Deißenbock과 Pizka[2]의 연구의 단점을 개선하기 위하여 식별자 구성 패턴과 WordNet [7]을 활용하였다. 동음 이의어의 경우 보통 식별자 내에 또 다른 식별자가 존재한다는 패턴을 기반으로 식별하였다. 예를 들면, FileName과 FileHandle내에 모두 File을 가지는 경우가 그 예이다. 또한, 동의어를 식별

하기 위하여 의미상 유사한 단어정보를 제공하는 온톨로지인 WordNet을 활용하여 동의어 검색을 자동화 하였다. 하지만, 여기에서는 단어의 품사를 분석하지 않았기 때문에 동의어의 범위는 상당히 넓고, 의미의 유사 정도도 매우 상이할 수 있어 정확도는 많이 떨어지게 된다.

Abebe et al.[4]는 Lexicon Bad Smell이라는 적절치 않은 식별자를 지칭하는 개념을 소개하고, 이는 향후 리팩토링[19]의 대상이 될 수 있다고 소개하였다. 여기에서는 잘못된 식별자의 증상과 어떻게 리팩토링을 해야 하는지에 대한 소개와 함께 이를 자동으로 탐지할 수 있는 도구인 LBSDetector를 소개하였다. LBSDetector를 사용하면 Class의 이름에 명사가 포함되었는지, 또는Class식별자와 동일하거나 Class 식별자를 포함하고 있는 Attribute, Operation 식별자가 있는지 등을 파악할 수 있다. 하지만, 이 접근 방법은 비 일관적 식별자는 동음 이의어에 대해서만 다루었으며, 이는 Lawrie et al.[3]의 접근 방법과 유사하다. 또한, 검출 방법이 엄격하여 개발자에게 오히려 불편을 줄 수도 있다.

Abebe와 Tonella[5]과 Falleri et al.[6]은 소스 코드의 식별자를 사용하여 개념과 개념 사이의 관계를 포함한 온톨로지를 구축하였다. 여기에서는 코드 식별자를 구성하는 단어로 분리한 후, 제안한 규칙에 따라 문장을 생성하여 자연어 파서를 통하여 단어간의 관계를 파악하고, 이를 지식화하였다. 이를 통하여 개발자들은 코드에 사용된 단어와 관련 지식을 검색하여 식별자를 명명할 수 있다. 이 연구는 자연어 파서를 사용하여 소스 코드를 분석한다는 점에서 접근 방식은 비슷하나, 연구의 목적이 비 일관성 식별자를 검출하는 것이 아닌 코드 구성 개념간의 관계(온톨로지)를 생성하기 위한다는 점에서 다르다.

6. 결 론

본 연구에서는 소스 코드 내의 비 일관 식별자를 검출하기 위하여 자연어처리 기법을 적용하였다. 본 연구에서는 비 일관성에 대하여 의미상 유사어 비일관성, 구조상 유사어 비 일관성, 구문상 품사 비 일관성, 단어 품사 비 일관성으로 구분하였다. 이를 자동으로 검출하기 위하여 자연어 구문 분석기인 Stanford Parser와 WordNet을 활용하였고, Levenshtein Distance 알고리즘을 적용하였다. 또한, CodeAmigo라는 지원 도구를 소개하였으며, 오픈 소스 프로젝트인 Apache Lucene과 Apache Ant에 본 접근 방법을 적용하여 본 연구의 타당성을 보였다.

참고문헌

[1] N. Madani, L. Guerroju, M.D. Penta, Y. Gueheneuc and G. Antoniol, “Recognizing Words from Source Code Identifiers using Speech Recognition Techniques”, In Proceedings of 14th European Conference on Software Maintenance and Reengineering(CSMR), Madrid, Spain, 2010, pp. 68-77.

[2] F. Deibenbock and M. Pizka, “Concise and Consistent Naming”, In Proceedings of International Workshop on Program Comprehension 2005(IWPC 2005), St. Louis, MO, USA, 2005, pp.261-282.

[3] D. Lawrie, H. Field and D. Binkley, “Syntactic Identifier Conciseness and Consistency”, In Proceedings of Sixth IEEE International Workshop on Source Code Analysis and Manipulation(SCAM2006), Philadelphia, Pennsylvania, USA, Sept 2006, pp.139-148.

[4] S.F. Abebe, S. Haiduc, P. Tonella and A. Marcus, “Lexicon Bad Smells in Software”, In Proceedings of 16th Working Conference on Reverse Engineering, Antwerp Belgium, Oct 2008, pp.95-99.

[5] S.L. Abebe and P. Tonella, “Natural Language Parsing of Program Element Names for Concept Extraction”, In Proceedings of 18th International Conference on Program Comprehension (ICPC 2010), Braga, Minho, Portugal, July 2010, pp.156-159.

[6] J. Falleri, M. Lafourcade, C. Nebut, V. Prince and M. Dao, “Automatic Extraction of a WordNet-like Identifier Network from Software”, In Proceedings of 18th International Conference on Program Comprehension (ICPC 2010), Braga, Minho, Portugal , July 2010, pp.4-13.

[7] WordNet: A lexical database for English, Home page (2012), http://wordnet.princeton.edu/

[8] D. Klein and C.D. Manning, “Accurate Unlexicalized Parsing”, In Proceedings of the 41st Meeting of the Association for Computational Linguistics, Sapporo, Japan, 2003, pp. 423-430.

[9] V.I Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals”, Soviet Physics Doklady, vol. 10, no. 8, pp. 707-710, 1966.

[10] “Code Conventions for the Java Programming Language: Why Have Code Conventions”, Sunmicro Systems (1999), Available: http://www.oracle.com/technetwork/java/index-135089.html

[11] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia and E. Merlo, “Recovering Traceability links between code and documentation”, IEEE Transactions on Software

Engineering, vol. 28, no.10, pp.970-983, October 2012.
[12] B. Caprile and P. Tonella, “Nomen Est Omen: Analyzing the Language of Function Identifiers”, In Proceedings of Sixth Working Conference on Reverse Engineering, Altanta, Georgia, 1999, pp.112-122.
[13] The Stanford Parser Home page, 2012, Available: http://nlp.stanford.edu/software/lex-parser.shtml
[14] JAWS(Java API for WordNet Searching) Homepage, 2012, Available: http://lyle.smu.edu/~tspell/jaws/index.html
[15] JWI(The MIT Java WordNet Interface) Homepage, 2012, Available: http://projects.csail.mit.edu/jwi/
[16] HyperSQL Homepage, 2012, Available: http://www.hsqldb.org/
[17] Apache Lucene Homegage, 2012, Available: http://lucene.apache.org/core/
[18] Apache Ant Homepage, 2012, Available: http://ant.apache.org/
[19] M. Fowler, “Refactoring: Improving the Design of Existing Code”, Addison-Wesley, 1999.

약 력

김 순 태

2003 중앙대 컴퓨터공학과 학사
2007 서강대 컴퓨터공학과 석사
2010 서강대 컴퓨터공학과 박사
2003~2005 (주)JKD Soft. 전임 연구원
2011~현재 강원대학교 컴퓨터과학과 교수
관심분야 : 소프트웨어공학, 소프트웨어 아키텍처, 디자인 패턴, Mining Software Repository
E-mail : stkim@kangwon.ac.kr

심 빈 구

2001 중앙대학교 컴퓨터공학과 졸업(학사)
2001~2003 (주) JKDSoft 아키텍트
2004~2007 (주) 소프트웨어크래프트, 선임 컨설턴트
2007~2009 서강대학교 컴퓨터학과 졸업(석사)
2009~2011 (주) 드리머 Product팀 팀장
2011~현재 소프트웨어 엑스퍼트 그룹 컨설턴트
관심분야 : Empirical software engineering, Software repository mining, 아키텍처
E-mail : lanore@swexpertgroup.com