MDD 기법을 이용하여 생성된 코드 간의 기능적 유사도 및 코드 생성률 측정 기법

류성태, 박철현, 이은석
성균관대학교 전자전기컴퓨터공학과
e-mail : xenoz0718@gmail.com, pch851130@naver.com, eslee@ece.skku.ac.kr

Measuring Methods of Functional Similarity and Code Generation Rate for the Code Generated by MDD

Sung-tae Ryu, Chul-hyun Park, Eunseok Lee
Dept. of Electrical and Computer Engineering, Sungkyunkwan University

요 약

오늘날 모바일 시장을 중심으로 다양한 플랫폼이 등장하면서 모바일 어플리케이션 개발 시 여러 플랫폼을 고려해야 하는 부담이 증대되고 있다. 이러한 상황에서 Model-Driven Development (MDD) 는 멀티플랫폼에 대응하는 어플리케이션 개발의 효율성을 높여줄 수 있다. 하지만 이 기법을 이용하는 대다수의 연구 결과들은 해당 방법론을 통해 생성된 결과물의 질을 객관적으로 평가할 수 없고, 이 때문에 해당 방법론의 성능 평가가 힘들다. 본 연구에서는 대상 플랫폼들이 제공하는 API 를 분석한 결과에 근거하여 공통 요소를 추출하고 이를 이용하여 MDD 기반으로 개발을 진행할 수 있는 개발 프로세스를 소개하고, 이를 통해 생성된 소스 코드의 기능적 유사도 및 코드 생성률과 기능적 유사도를 평가할 수 있는 방법을 제안한다. 이 방법은 코드를 AST 로 바꾸고 API 맵핑 테이블에 근거하여 동일한 키워드로 변환하고 유사도를 측정하여 설계 시 의도한 기능이 얼마나 잘 코드로 생성되었는 지 평가할 수 있는 방법이다. 본 연구에서는 이 방법을 이용하여 생성된 코드의 기능적 유사도와 코드 생성률을 측정하였다.

1. 서론

최근 모바일 시장을 중심으로 다양한 기기와 플랫폼이 등장하고 있다. iPhone 과 Android 가 강력한 양강체제를 구축한 가운데, Windows Phone, BADA, Symbian, LiMo 등의 다양한 플랫폼들이 등장하여 경쟁이 심화되고 있다. 이 때문에 개발자들은 개발 시 여러 플랫폼에 동시에 대응해야 하는 어려움을 겪고 있다. 개발자들은 복수의 플랫폼에 대응하기 위해 서로 다른 개발 환경과 프로그래밍 언어에 익숙해져야 한다. 특히 동일한 기능을 하는 어플리케이션을 각 플랫폼에 맞게 따로 개발해야 하여 추가 개발 비용 및 시간을 부담해야 하는 실정이다.

이러한 상황에서 Model-Driven Development (MDD)는 하나의 해결책이 될 수 있다. 현재 모바일 어플리케이션의 UI 를 중심으로 API 의 기능적인 유사도 분석을 통해 동시에 복수의 플랫폼에서 모바일 어플리케이션을 개발할 수 있는 연구[3]가 진행되고 있다.

하지만 기존의 MDD 관련 연구들은 그 성능을 측정할 수 있는 객관적인 지표의 부재로 인해 그 우수성을 평가하기 어렵다. 특히 MDD 를 적용하여 하나의 모델을 변환하여 생성된 소스 코드에 설계자가 의도한 기능이 얼마나 정확하게 반영되어 있는 지에 대해 평가하기 어렵다.

본 연구에서는 이러한 문제를 해결하기 위해 생성된 코드를 AST 로 변환하여 비교하는 방법을 제시하고 실제로 서로 다른 두 소스 코드가 어느 정도로 유사한 기능을 하는 지 제안 방법을 이용하여 평가한다. 또한 하나의 통합 모델을 코드를 생성하였을 때 설계 단계에서 의도한 데로 소스 코드가 정확히 생성되는 지에 대해 코드 생성률이라는 지표를 제시한다.

2. 관련연구

기존에도 AST 를 이용하여 코드를 비교하는 방법과 관련한 많은 연구들이 있었다. Higo[4]는 소프트웨어의 유지보수 측면에서 코드 클론을 AST 를 이용하여 시각화하는 연구를 제안하였다. D.Ballis[5]는 UML 모델에서 디자인 패턴을 찾기 위해 AST 를 이용한 Rule-Based 방법을 제안하였다. Beat[2]는 소스 코드의 버전 간의 세밀한 변경 사항을 찾아내기 위해 AST 를 이용하는 방법을 제안하였다. 이러한 연구들은 대부분 소프트웨어 개발 프로세스에서 소스 코드의 변경 부분을 효율적으로 찾아내는 데 그 목적을 두고 있다. 이 때문에 동일한 언어로 작성된 소스 코드 간의 비교만이 가능하다는 점과, 기능적인 변화보다는 구조적인 변화를 찾는 것에 초점을 맞추고 있다.

* 이 논문은 2009 년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. 2010-0016585)

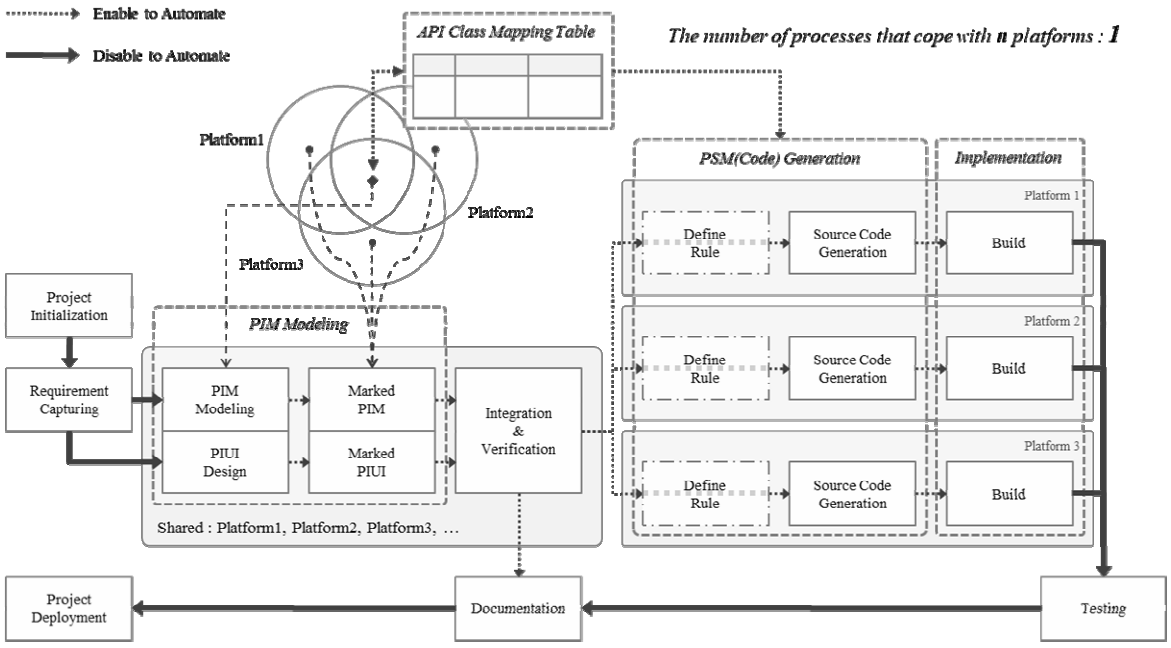

그림 1. 반-단일 개발 프로세스

본 연구에서는 ANTLR[9]를 이용하여 소스 코드를 XML 형식의 AST 로 변환하는 파서를 생성한다. ANTLR 는 언어 인식을 위한 오픈 소스 프레임워크이다. 이것은 인식기, 해석기, 컴파일러, 번역기 등을 위한 프레임워크를 제공한다. 또한 변환된 코드를 비교하는 단계에서는 API 맵핑 테이블을 이용하여 Beat[2]의 Changing Distilling 기법을 개선시켜 사용한다.

3. 반-단일 개발 프로세스 (HDSP)

본 연구는 반-단일 개발 프로세스(HSDP : Half-Single Development Process)[3]에서 생성된 소스 코드 간의 기능적 유사도와 코드의 생성률을 측정하기 위한 방법을 제안하고 있다. 따라서 본 연구의 성과를 이해하기에 앞서 이 장에서는 제안하는 방법을 설명하기 앞서 HSDP 에 대해 개략적으로 소개한다.

HSDP 은 MDD 를 적용하여 복수의 플랫폼을 대상으로 동일한 요구사항이나 목적을 가지는 어플리케이션을 개발할 때 개발 효율성을 높일 수 있는 프로세스이다. 코드 생성 직전 단계까지 개발 프로세스를 공유되며, 소스 코드를 생성하는 단계에서야 대상으로 하는 플랫폼의 수에 맞추어 프로세스가 나뉜다.

그림 1 에서 보는 것과 같이 *Platform Independent Model (PIM) Modeling* 단계에서는 대상 플랫폼들의 공통 요소를 반영하여 통합 모델을 설계한다. *Marked PIM* 은 특정 플랫폼에서만 동작하는 요소가 추가적으로 서술된 결과이다. 코드 생성 규칙은 각 플랫폼 별로 서술되는 것으로서 모델의 각 요소를 어떻게 해석하고, 이를 어떤 코드로 생성할 것인 지에 대한 내용을 담고 있는 재사용 가능한 문서이다.

재사용이 가능한 코드 생성 규칙 때문에 *PIM Modeling* 단계 이후의 개발은 대부분 자동화될 수 있으며, 개발 중인 프로그램의 요구 사항 변경이나 유지 보수 등의 상황이 발생하면 *PIM Modeling* 단계만 다시 수행하는 것으로 문제를 해결 할 수 있다.

하나의 서술 내용을 통해 서로 다른 플랫폼에서 동일한 기능을 하는 코드를 효율적으로 생성해내기 위해서는 각 플랫폼에서 제공하는 API 의 공통점과 차이점을 분석할 필요가 있다. 이는 API 맵핑 테이블로 기술되며 이것은 개발할 때마다 재참조 가능한 독립적인 자원이다.

다음은 HSDP 의 특징과 장점을 요약한 것이다:

- 코드 생성 규칙은 처음에만 정의하고 필요에 따라 보완하면 재사용이 가능하다. 따라서 이후 *Requirement Capturing* 과 *PIM Modeling* 을 제외한 모든 개발 과정은 자동화될 수 있다.
- 다수의 플랫폼에 대응하는 어플리케이션을 개발하더라도 하나의 *PIM* 만 모델링하고 서술하면 된다. 이 때문에 플랫폼 별로 대응하는 개발 프로세스는 최대한 공유될 수 있다.
- 플랫폼들의 프레임워크를 분석하여 공통점을 *PIM* 에 반영하고 차이점을 *Marked PIM* 에 반영한다. 코드 생성 규칙에서는 이를 판별하는 내용을 서술하여 대상 플랫폼이 지원하지 않는 기능을 *Wrapper Class* 와 같은 추가적인 소스 코드를 생성함으로써 플랫폼간의 차이점을 최소화할 수 있다.

4. API 맵핑 테이블을 이용한 Change Distilling 기법

Change Distilling 기법은 소프트웨어 개발 진행 및 유지 보수 시에 각 소스 코드를 리비전 간의 변경사항을 알아내는 데 유용하다. 본 연구에서는 서로 다른 프로그래밍 언어로 된 소스 코드간의 기능적 유사도를 비교할 필요가 있기 때문에 해당 기법을 그대로 사용하기에는 무리가 있다. 본 연구에서는 서로 다른 프로그래밍 언어와 플랫폼을 대상으로 작성된 소스 코드 간의 비교라는 목적에 맞게 *Change Distilling* 기

법을 개선시켜 사용한다.

Beat[2] 은 소스 코드에서 변화를 추출하기 위해 트리 구조의 차이점을 수치적으로 측정하는 *Change Distilling* 기법을 제안하였다. 이 기법은 *n-Gram String Similarity* 라는 강인한 문자열 비교 기법을 이용하여 소프트웨어 개발 과정에서 소스코드의 두 리비전 사이의 변화를 발견해낸다.

$$sim_{ng}(s_1, s_2) = \frac{2 \times |n\text{-}gram(s_1) \cap n\text{-}gram(s_2)|}{|n\text{-}gram(s_1)| + |n\text{-}gram(s_2)|}$$

소스 코드를 구성하는 문장의 특성상 아래의 경우와 같이 *verticalDrawAction* 과 *drawVerticalAction* 은 동일하다고 판정해야 하는 경우가 발생할 수 있다. 이 때는 *Bi-Gram* 비교 기법이 가장 유사하다고 판정하기 때문에 *Change Distilling* 기법에서는 이 비교 기법을 이용하여 각 트리 요소를 비교한다.

s_1 = verticalDrawAction $\quad$ s_2 = drawVerticalAction

$$sim_{2g}(s_1, s_2) = \frac{2 \times 14}{34} \approx 0.82 \quad sim_{3g}(s_1, s_2) = \frac{2 \times 12}{34} \approx 0.75$$

서로 다른 플랫폼을 대상으로 작성된 두 소스 코드의 유사도를 비교하기 위해 각 API 의 클래스명 또는 메소드명, 속성명 등을 동일한 키워드로 바꿔줄 필요가 있다. 아래는 특정 노드의 값이 *Bi-Gram* 비교 기법으로 특정 유사도 이상으로 API 맵핑 테이블에서 발견될 경우 해당 키워드로 변경한다는 기준이다. 즉, 그림 2 에서 보는 바와 같이 AST 로 변환된 소스 코드의 각 요소는 API 맵핑 테이블을 참조하여 동일한 요소로 인식될 수 있도록 동일한 키워드로 변경되어 비교된다.

$$map(x) = \begin{cases} v(y_{table}) & \text{if } sim_{2g}(v(x), v(y_{table})) \geq f \\ v(x) & \text{otherwise} \end{cases}$$

동일한 키워드로 변경된 AST 를 매칭하는 첫번째 기준은 Leaf 노드들을 *Bi-Gram* 측정 기법을 이용하여 기준 값 이상 유사하다고 판단될 경우 동일한 노드로 판정하는 것이다. 동일한 노드로 판명된 하나의 노드를 발견하더라도 분석이 중지되지는 않으며 전체 노드를 비교하여 최고의 유사도를 보이는 노드를 찾는다.

$$match_1(x, y) = \begin{cases} true & \text{if } l(x) = l(y) \wedge sim_{2g}(v(x), v(y)) \geq f \\ false & \text{otherwise} \end{cases}$$

두번째 매칭 기준은 Inner-Node 들을 *Bi-Gram* 비교 기법을 이용하여 그 유사도를 측정하는 방법이다. 단계에서는 두 노드가 공통으로 가지는 서브 트리가 있는 지 비교하여 노드의 값뿐만 아니라 자식 노드들로 구성된 서브 트리 구조의 유사도까지 측정한다.

$$match_2(x, y) = \begin{cases} true & \text{if } l(x) = l(y) \wedge \frac{|common(x, y)|}{max(|x|, |y|)} \geq t \\ & \wedge sim_{2g}(v(x), v(y)) \geq f \\ false & \text{otherwise} \end{cases}$$

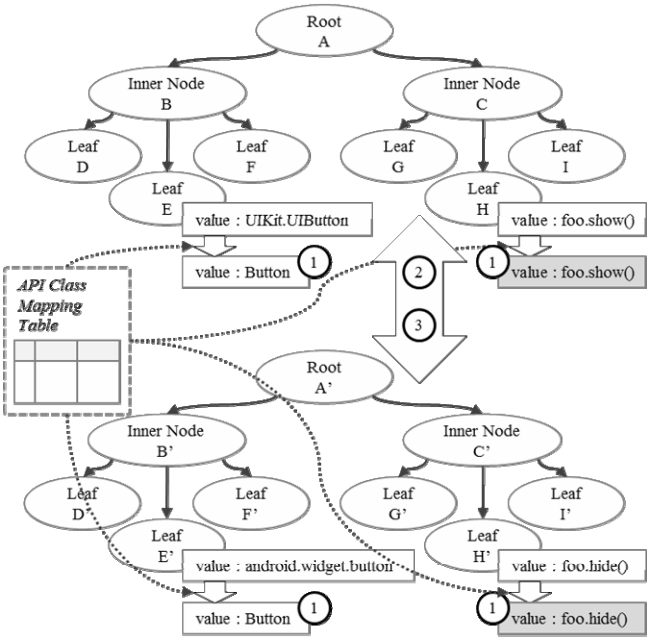

그림 2. API 맵핑 테이블을 참조하여 코드의 AST 를 비교

5. 코드 생성률 및 기능적 유사도 측정 방법

코드 생성률은 PIM 의 AST 와 생성된 코드의 AST 를 이용하여 코드가 설계 단계에서 의도한 데로 얼마나 정확하게 생성되었는 지를 측정하는 지표이다. 이것을 통해 본 연구에서 제안하는 HSDP 의 코드 생성률을 평가할 수 있다. 또한 다양하게 존재할 수 있는 코드 생성 규칙의 성능을 나타내는 하나의 지표로 사용될 수 있다.

$$CGR(T_{\text{Model}}, T_{\text{Generated}}) = \frac{|nodes(T_{\text{Model}}) \cap nodes(T_{\text{Generated}})|}{|nodes(T_{\text{Model}})|}$$

$$nodes(T) = \{x \mid x = map(a), a \in T\}$$

$$nodes(T_1) \cap nodes(T_2) = \{x, y \mid x \in T_1, y \in T_2, match(x, y) = true\}$$

$$nodes(T_1) \cup nodes(T_2) = \{x, y \mid x \in nodes(T_1), y \in nodes(T_2)\}$$

기능적 유사도는 하나의 PIM 모델을 기반으로 생성된 서로 다른 프로그래밍 언어로 된 두 소스 코드를 AST 로 변환하여 그 유사도를 비교하는 것이다. 이 둘의 유사도가 높을 수록 HSDP 가 하나의 모델을 동일한 기능을 하는 서로 다른 플랫폼에서 동작하는 소스 코드를 잘 생성해냈다고 판단할 수 있다. 코드 생성률처럼 직접적인 지표는 아니지만, HSDP 의 성능을 간접적으로 나타낼 수 있는 하나의 지표이다.

$$FS(T_{\text{Code1}}, T_{\text{Code2}}) = \frac{|nodes(T_{\text{Code1}}) \cap nodes(T_{\text{Code2}})|}{\max(|nodes(T_{\text{Code1}})|, |nodes(T_{\text{Code2}})|)}$$

6. 사례 연구

본 연구에서 제안하는 방법을 평가하기 위해 총 8 가지의 사례를 준비하였다. 각 사례는 XML 에 기반한 하나의 PIM 모델과 이를 기반으로 생성된 iPhone 에서 동작하는 Objective-C 로 작성된 소스 코드, Android

<표 1> 동일한 기능을 목적으로 생성된 소스 코드 간의 기능적 유사도 및 코드 생성률 측정

테스트 케이스		1	2	3	4
	코드의 기능적 유사도	0.286	0.200	0.667	0.385
API 맵핑 테이블 사용	코드의 기능적 유사도	0.571	0.267	1.000	0.615
	코드 생성률 (Objective-C)	0.875	0.833	1.000	0.857
	코드 생성률 (JAVA)	1.000	0.950	1.000	0.857

<표 2> 서로 다른 기능을 목적으로 작성된 소스 코드 간의 기능적 유사도 및 코드 생성률 측정

테스트 케이스		1	2	3	4
API 맵핑 테이블 사용	코드의 기능적 유사도	0.075	0.059	0.098	0.064

에서 동작하도록 Java 로 작성된 소스 코드를 하나의 그룹으로 하여 구성되어 있다. 각 PIM 모델은 하나의 메소드 단위로 서술된 모델이며, 생성된 소스 코드 역시 하나의 메소드이다.

생성된 소스 코드들을 ANTLR 로 개발한 파서를 이용하여 XML 기반의 AST 로 변환한 뒤 제안 기법을 이용하여 그 유사도를 평가하였다. 먼저 동일한 동작과 기능을 목적으로 하는 소스 코드 4 쌍을 비교하였다. 그 결과는 표 1 에서 보는 바와 같다. 또한 서로 다른 동작과 기능을 목적으로 하는 소스 코드 4 쌍도 비교하였다. 그 결과는 표 2 에서 보는 바와 같다.

4 가지 사례 모두 생성된 2 종류의 소스코드에 대한 코드 생성률이 높은 것으로 나타났으며 이는 의도한 모델이 소스 코드로 잘 생성되었다는 것을 의미 한다. 또한 코드의 기능적 유사도 또한 API 맵핑 테이블을 사용한 경우, 그렇지 않은 경우보다 높은 수치를 나타냈다. API 맵핑 테이블을 좀 더 세분화하여 정리하고 각 프로그래밍 언어의 문법적인 특징을 반영하여 알고리즘을 개선한다면 좀 더 정확한 수치가 나올 것으로 기대된다.

7. 결론

MDD 프레임워크의 성능을 평가하기 위한 핵심적인 이슈는 소스 코드가 설계 시 의도한 데로 얼마나 정확하게 생성되었는 지 확인하는 것이다. 하지만 지금까지는 MDD 프레임워크를 검증하기 위한 적당한 방법이 존재하지 않았다. 본 연구에서는 MDD 프레임워크의 성능을 평가할 때 발생하는 어려움을 극복하기 위해 코드 생성률이라는 개념을 제안하였다.

이것을 측정하기 위해 본 연구에서는 Change Distilling 의 API 맵핑 테이블에 근거하여 향상시킨 수정 버전을 작성하여 사용하였고, 이를 통해 서로 다른 프로그래밍 언어로 작성된 코드에서 기능적인 유사도 및 구조적인 유사도를 측정할 수 있었다. 제안하는 방법은 MDD 프레임워크의 성능을 객관적으로 평가해줄 뿐만 아니라 다양하게 존재할 수 있는 코드 생성 규칙의 개별적인 성능 평가를 가능하게 해준다.

하지만 이 방법은 크게 3 가지 약점이 있다. 첫 번째는 소스 코드를 메소드나 함수 단위로 비교해야 한다는 것이다. 두 번째 약점은 API 맵핑 테이블에 의존적이라는 것이다. 정확도의 향상을 위해서는 더 자세하고 세밀한 API 맵핑 테이블의 서술이 필요하며 이것을 작성하는 것은 많은 시간을 요구한다. 마지막 약점은 하나의 모델로부터 생성된 소스 코드간의 비교만이 가능하다는 점이다. 이 때문에 우리는 같은 기능을 의도하였지만 서로 다른 설계에 기반하고 있는 코드 사이에서는 정확한 결과를 기대할 수 없다.

향후에는 이러한 단점을 보완하고, 코드 생성률 및 기능적 유사도 측정 기능을 개발중인 HSDP 지원 프레임워크에 적용하여 개발 시 실시간으로 확인할 수 있도록 지원할 계획이다.

참고문헌

[1] Vasian Cepa, Mira Mezini, "Language support for model-driven software development", Science of Computer Programming, pp.13-25, Elsevier, 2008.

[2] Beat Fluri, Martin Pinzger, "Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction", IEEE Transaction on Software Engineering, Vol33. No.11, pp.725-743, IEEE, 2007

[3] Sung-tae Ryu, Cheolhyun Park, Hyunsang Youn, Eun-seok Lee, "Model-driven Development Approach to Generating Mobile Application Source Code for Graphic User Interface," ICHIT 2010, pp. 353-360, August, 2010

[4] Yoshiki Higo, Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue, "Method and implementation for investigating code clones in a software system", Information and Software Technology, pp.985-998, 2006

[5] D. Ballis, A. Baruzzo, M. Comini, "A Rule-based Method to Match Software Patterns Against UML Models", Electronic Notes in Theoretical Computer Science, pp.51-66, 2008

[6] Robert L. Akers, Ira D. Baxter, Michael Mehlich, Brian J. Ellis, Kenn R. Luecke, "Case Study: Re-engineering C++ component models via automatic program transformation", Information and Software Technology, pp.275-291, 2006

[7] Malte Appeltauer and Gunter Kniesel, "Towards Concrete Syntax Patterns for Logic-based Transformation Rules", Electronics Notes in Theoretical Computer Science, pp.113-132, 2008

[8] S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, “Change Detection in Hierarchically Structured Information,” Proc. ACM Sigmod Int’l Conf. Management of Data, pp. 493-504, June, 1996

[9] ANTLR, http://www.antlr.org