Journal of KIIT. Vol. 17, No. 1, pp. 141-148, Jan. 31, 2019. pISSN 1598-8619, eISSN 2093-7571 141
http://dx.doi.org/10.14801/jkiit.2019.17.1.141

모바일 애플리케이션의 전력 소비 감소를 위한 에너지 리팩토링 플러그인 설계 및 구현

김성보*, 최기현**, 구자환***, 김응모****

Design and Implementation of Energy Refactoring Plug-In for Reducing Power Consumption of Mobile Applications

Seong-Bo Kim*, Kee-Hyun Choi**, Ja-Hwan Koo***, and Ung-Mo Kim****

본 연구는 문화체육관광부 및 한국저작권위원회의 2018년도 저작권기술개발사업의 연구결과로 수행되었음

요 약

스마트폰 이용자들은 스마트폰을 사용할 때 불편한 점 중 하나로 배터리 용량의 부족을 꼽는다. 배터리 용량이 제한적인 모바일 기기의 특성으로 인해, 모바일 애플리케이션을 개발하는 과정에서 해당 애플리케이션이 실행될 때의 전력 소비를 최소화하도록 하여 더 오랫동안 서비스할 수 있게 만드는 것이 매우 중요하다. 본 논문에서는 안드로이드 스튜디오에서 모바일 애플리케이션을 개발할 때 전력 소모량의 감축을 위한 코드 리팩토링을 수행하는 플러그인을 설계 및 구현하였고, 이를 사용하여 샘플 모바일 애플리케이션의 소스 코드에 Encapsulate Field 기법을 적용한 결과 약 4.7%의 전력 효율 향상을 보였으며 이것은 실제 모바일 애플리케이션의 전력 소모량 감소에 효과가 있음을 의미한다.

Abstract

One of the inconveniences when using smart phones is the lack of battery capacity for smartphone users. Due to the limited battery capacity of the mobile device, it is very important to develop a mobile application so that it could be serviced for a longer period of time by minimizing power consumption when the application is run. In this paper, we designed and implemented a plug-in for code refactoring to reduce power consumption when developing a mobile application using the Android Studio. Using this, we applied the encapsulate field technique to the source code of the sample mobile application, showing about 4.7% power efficiency improvement. This means that there is an effect on the power consumption.

Keywords

energy refactoring, power consumption, android studio, plug-in, mobile application

* 성균관대학교 정보통신대학 석사과정
- ORCID: http://orcid.org/0000-0001-9601-6346
** 성균관대학교 정보통신대학 연구교수
- ORCID: http://orcid.org/0000-0001-9476-6389
*** 성균관대학교 사회과학대학 연구교수(교신저자)
- ORCID: http://orcid.org/0000-0002-2844-3183
**** 성균관대학교 소프트웨어대학 교수
- ORCID: http://orcid.org/0000-0001-5464-6358

· Received: Nov. 12, 2018, Revised: Jan. 15, 2019 Accepted: Jan. 18, 2019
· Corresponding Author: Ja-Hwan Koo
Room# 27309, Engineering bldg. 2, SungKyunKwan University, 2066 Seobu-Ro, Jangan-gu, Suwon-si, Gyeonggi-do 16419, South Korea
Tel.: +82-31-290-7218, E-mail: jhkoo@skku.edu

Ⅰ. 서 론

스마트폰 시장은 지난 10년간 엄청난 성장을 이룩해왔고, 오늘날 스마트폰은 수많은 사람에게 일상생활에서 빼놓을 수 없는 구성 요소가 되었다[1][2]. 그러나 스마트폰 사용자들은 여전히 부족한 배터리 수명에 불편함을 호소하고 있다[3]. 따라서 스마트폰 어플리케이션 개발자들은 스마트폰의 배터리 소모량을 가급적 줄일 수 있는 방향으로 코드를 작성하는 것이 매우 중요하다고 볼 수 있다[4]. 코드 리팩토링은 소스 코드의 구조를 개선함으로써 소프트웨어 시스템의 기능적 동작을 보장하며 성능 저하의 근본 원인을 확인하고 소프트웨어의 성능을 향상하는 프로세스이다. 이때, 전력 낭비를 초래하는 코드 패턴을 개선함으로써 전력 효율을 향상하는 리팩토링 기법을 에너지 리팩토링이라 한다[5]. 본 연구에서는 에너지 리팩토링을 수행하는 안드로이드 스튜디오 플러그인을 설계 및 구현하여 전력 효율이 더 높은 안드로이드 애플리케이션을 개발하는 데 도움이 되도록 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 연구에 관하여 기술하고, 3장에서는 구체적인 구현 방안을 보인다. 4장에서는 3장에서 설계한 기능에 대한 실제 성능을 평가하며, 5장에서는 결론을 기술하고 본 연구의 한계점 및 향후 개선 방안을 제시한다.

Ⅱ. 관련 연구

논문 [6]과 [7]에서 안드로이드 모바일 애플리케이션의 소스 코드를 자동으로 리팩토링하여 전력 효율을 향상하는 도구 Leafactor에 대하여 다루고 있다. 그러나 이는 2014년 10월부터 구글의 안드로이드 스튜디오가 안드로이드 앱 개발에 있어서 공식 IDE로 결정되고 Eclipse ADT 플러그인의 지원이 전면 중단된 상황에서, 아직 Eclipse가 필요한 시스템이라는 단점이 존재한다.

논문 [8]은 M. Fowler가 제시한 68개의 코드 리팩토링 기법[9]들에 대해 소모 전력량의 변화를 관찰하여 전력 효율을 측정 및 분석하였다. 이때, 전력 효율이 높다는 의미는 동일한 서비스를 제공하기 위해 더 적은 양의 전력을 소비하는 정도로 정의하였다. 소모되는 전력량의 차이가 얼마나 되는지 확인하기 위해 소프트웨어 전력 분석 도구인 XEEMU[10]을 사용하였다. M. Fowler가 제시한 예제를 기반으로 한 소스 코드를 통해 실험하여 선정한 전력 효율이 높은 리팩토링 기법은 총 26가지이며, 표 1과 같다.

논문 [8]에서 소프트웨어 개발자들은 위 결과를 통해 어떤 리팩토링 기법이 전력 효율성을 향상하는지 손쉽게 받아들일 수 있으며, 앞으로 전력 소모 측면을 고려하여 소프트웨어를 개발할 수 있을 것으로 전망하였다.

표 1. 전력 효율이 좋은 코드 리팩토링 기법
Table 1. Code refactoring techniques for energy efficiency

No.	Code Refactoring Techniques
R2	Inline Method
R3	Inline Temp
R5	Introduce Explaining Variable
R10	Move Method
R11	Move Field
R13	Inline Class
R15	Remove Middle man
R16	Introduce Foreign Method
R21	Change Reference to Value
R25	Encapsulate Field
R26	Encapsulate Collection
R29	Replace Subclass with Fields
R32	Consolidate Duplicate Conditional Fragments
R33	Remove Control Flag
R36	Introduce Null Object
R40	Remove Parameter
R41	Separate Query from Modifier
R42	Parameterize Method
R45	Replace Parameter with Method
R46	Introduce Parameter Object
R47	Remove Setting Method
R52	Pull Up Field
R53	Pull Up Method
R36	Push Down Field
R60	Collapse Hierarchy
R63	Replace Delegation with Inheritance

III. 안드로이드 에너지 리팩토링 플러그인 설계 및 구현

3.1 플러그인 개발 환경 설정

안드로이드 스튜디오가 IntelliJ IDEA 오픈소스를 이용하여 개발된 도구이기 때문에 안드로이드 스튜디오 플러그인의 개발 환경으로 IntelliJ IDEA가 필요하다. 안드로이드 스튜디오는 본 연구의 수행 시점 기준으로 2017.8.4. 버전의 IntelliJ IDEA로 개발된 도구이기 때문에 호환성 문제를 해결하기 위해 최신 버전이 아닌 이전 버전의 IntelliJ IDEA로 플러그인을 개발하였다.

안드로이드 스튜디오에서와 같이 IntelliJ IDEA는 자동으로 Gradle을 이용해서 프로젝트를 개발할 수 있도록 기능을 제공한다. 따라서 Gradle을 이용하여 개발하기 위해 최신 버전의 Gradle을 설치하였다.

예제 플러그인의 개발 절차는 plugin.xml 및 Action 클래스를 작성하고 플러그인을 실행한다는 점에서는 기본적인 프로젝트 개발과 유사하나, plugin.xml에서 클래스의 이름을 입력할 때 패키지명을 포함하여 완전한 클래스 이름을 입력해야 한다는 점을 유의해야 한다.

Action 클래스를 작성하기 위해서는 AnAction.java 클래스를 상속받아야 하며, plugin.xml의 <actions> 섹션에 표 2와 같이 태그를 추가함으로써 Action을 등록할 수 있다. 표 2의 예제 태그는 Action이 그룹 EditorPopupMenu에 등록된 상태일 때 컨텍스트 메뉴에서 Action을 사용하기 위해 추가한 것이다.

표 2. plugin.xml의 action 태그 예제
Table 2. Example of the action tag in plugin.xml

```
1  <actions>
2      <action id="EditorBasics.EditorIllustration"
3  class="EditorIllustration" text="Editor Basics"
4  description="Illustrateshow to plug an action in">
5      <add-to-group group-id="EditorPopupMenu"
   anchor="last"/>
6      </action>
7  </actions>
```

3.2 플러그인 참조 API

플러그인을 개발하기 위해서는 IntelliJ에서 제공하는 VFS(Virtual File System), Document, 편집기, PSI(Program Structure Interface), FVP(File View Provider) 등의 API에 대한 분석 및 설정이 필요하다[11].

VFS는 IntelliJ 플랫폼의 구성 요소로서 파일 작업을 위한 대부분의 활동을 캡슐화하며, 수정 사항이 발견될 때 파일 내용을 추적하고 이전 및 새 버전을 제공하는 기능과 추가 영구 데이터를 VFS의 파일과 연관시킬 가능성을 제공하는 기능을 수행한다.

Document는 일반적으로 가상 파일의 텍스트 내용에 해당하는 편집 가능한 유니코드 문자 시퀀스로서, Document를 통해서 파일 내용을 일반 텍스트 레벨로 액세스하거나 수정하는 작업을 수행할 수 있다. 일부 작업이 파일의 텍스트 내용에 액세스해야 할 때 문서 인스턴스가 작성된다. 특히 파일에 대한 PSI(Program Structure Interface)를 작성하는 데 필요하다. 또한, 대화 상자에서 텍스트 편집기 필드의 내용을 나타내기 위해 가상 파일에 링크되지 않은 문서 인스턴스를 임시로 만들 수 있다.

IntelliJ Platform 편집기를 이용해 텍스트를 처리하기 위해서는 update 메서드를 재정의함으로써 동작을 표시하고 실행할 수 있는 조건을 만들어야 한다. 이때, 프로젝트가 열려있는지, 사용할 수 있는 편집기의 인스턴스가 있는지, 선택된 텍스트 항목이 있는지를 확인해야 한다.

PSI는 IntelliJ 플랫폼의 파일 구문을 분석하고 플랫폼 기능을 지원하는 구문 및 의미 코드의 모델을 작성하는 일을 담당하는 계층이다. 단일 PSI 파일은 특정 프로그래밍 언어로 여러 개의 PSI 트리를 포함할 수 있다. 또한, PSI 요소는 하위 PSI 요소를 가질 수 있다. 개별 PSI 요소 수준의 PSI 요소 및 작업은 IntelliJ 플랫폼에서 해석되는 대로 소스 코드의 내부 구조를 탐색하는 데 사용된다. PSI 요소를 통하여 이름을 알고 있지만, 경로를 모르는 경우 파일을 찾거나, 특정 PSI 요소가 사용되는 위치를 찾거나, PSI 요소의 이름을 변경하거나, 가상 파일을 재구축하는 작업을 수행할 수 있다. JAVA 언어만

클래스의 모든 상속 클래스 찾기, 자격 있는 이름으로 클래스 찾기, 짧은 이름으로 클래스 찾기, 클래스의 상위 클래스 찾기, 클래스를 포함하는 패키지에 대한 참조 얻기, 특정 메서드를 override하는 메서드 찾기 등을 수행할 수 있다.

FVP는 단일 파일 내에서 여러 PSI 트리에 대한 액세스를 관리하는 역할을 한다. FVP를 통해서 파일에 PSI 트리가 있는 모든 언어 집합을 가져오거나, 특정 언어에 대한 PSI 트리를 가져오거나, 파일의 지정된 오프셋에서 특정 언어의 요소를 찾는 기능을 수행할 수 있다.

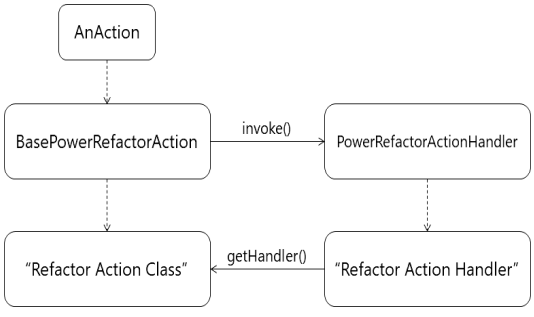

그림 1. 파워 리팩터 ERD
Fig. 1. Power refactor ERD

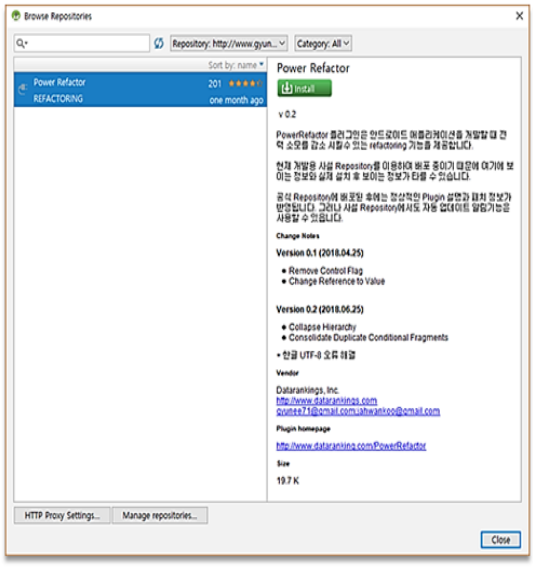

그림 2. 사설 Repository에 등록된 Power Refactor 플러그인
Fig. 2. Power refactor plug-in registered in private repository

3.3 플러그인 클래스 다이어그램

전력 소모를 감소하는 리팩토링 방법에 따라서 필요한 정보 및 요구 조건이 다르므로 통합적으로 관리하는 BaseRefactoringAction 클래스를 개발하고 각 리팩토링 방법에 맵핑되는 클래스는 BaseRefactoringAction에서 상속받아 처리한다. 그림 1에서 BasePowerRefactorAction은 각 리팩터의 동작 조건을 검사하고 필요한 정보를 로딩하는 상위 클래스로써 모든 파워 리팩터가 상속받는다. 각 리팩터 클래스는 자신의 기능을 수행할 핸들러 클래스를 작성하고 핸들러 클래스의 invoke 메소드를 통해서 각 리팩터가 동작하게 된다. 핸들러 클래스는 PowerRefactor ActionHandler 클래스를 상속받아 작성하며, 해당 클래스에 있는 invoke 메소드는 각 리팩터를 구현하는 클래스의 ActionHandler 클래스에서 구현해야 한다. 실제 PowerRefactorActionHandler는 인터페이스로 작성하였다.

그림 2는 개발된 플러그인이 사설 Repository에 등록된 것을 보여주고 있다.

3.4 플러그인 클래스 구현

본 연구를 통해 Inline Method, Inline Temp, Introduce Explaining Variable, Move Method, Move Field, Inline Class, Remove Middle Man, Encapsulate Field, Introduce Parameter Object, Pull Up Field, Pull Up Method, Push Down Field 등의 전력 효율이 좋은 리팩토링 기법을 일부 수정하여 개발하였다. 본 장에서는 이 중에서 표 3에 나와 있는 Encapsulate Field 기법에 대한 소스 코드를 예시로 들어 설명한다.

EncapsulateAction 클래스는 BasePowerRefactoringAction 상속된 자식 클래스로 현재 사용자가 에디터를 이용하고 있고 파일을 선택 중인지를 확인하여 해당 리팩토링 메뉴가 사용될 수 있도록 처리하는 여러 상속 메소드를 사용하거나 새로 구현하여 작성하였다.

RefactoringActionHandler는 BasePowerRefactoringAction에서 기본으로 처리되는 메소드를 처리하고 난 후에 실제 리팩토링을 적용한 메소드 핸들러를 호출 할 때 사용되는 메소드이다.

표 3. 안드로이드 에너지 리팩토링 플러그인의 EncapsulateAction 클래스
Table 3. EncapsulateAction class of android energy refactoring plugin

```
1  public class EncapsulateAction extends BasePowerRefactoringAction {
2      public boolean isAvailableInEditorOnly() {    return false;    }
3      @Override
4      protected boolean isAvailableOnElementInEditorAndFile(@NotNull PsiElement element, @NotNull Editor editor,
5  @NotNull PsiFile file, @NotNull DataContext context) {
6          final PsiElement psiElement = file.findElementAt(editor.getCaretModel().getOffset());
7          final PsiClass containingClass = PsiTreeUtil.getParentOfType(psiElement, PsiClass.class, false);
8          if (containingClass != null) {
9              final PsiField[] fields = containingClass.getFields();
10             for (PsiField field : fields) {
11                 if (isAcceptedField(field)) return true;
12             }
13         }
14         return false;
15     }
16     public boolean isEnabledOnElements(@NotNull PsiElement[] elements) {
17         if (elements.length == 1) {
18             return elements[0] instanceof PsiClass && elements[0].getLanguage().isKindOf(JavaLanguage.INSTANCE)
19 || isAcceptedField(elements[0]);
20         }
21         else if (elements.length > 1) {
22             for (int idx = 0; idx < elements.length; idx++) {
23                 if (!isAcceptedField(elements[idx])) {   return false;    }
24             }
25             return true;
26         }
27         return false;
28     }
29     public RefactoringActionHandler getHandler(@NotNull DataContext dataContext) {   return new
30 EncapsulateHandler();    }
31
32     private static boolean isAcceptedField(PsiElement element) {
33         return element instanceof PsiField && element.getLanguage().isKindOf(JavaLanguage.INSTANCE) &&
34 ((PsiField)element).getContainingClass() != null;
35     }
36 }
```

RefactoringActionHandler는 BasePowerRefactoring Action에서 기본으로 처리되는 메소드를 처리하고 난 후에 실제 리팩토링을 적용할 메소드 핸들러를 호출할 때 사용되는 메소드이다.

해당 리팩토링 메소드가 처리하지 못하는 것일 경우 경고 메시지를 출력하고 종료한다. 또한, 각 리팩토링 메소드는 핸들러를 정의해야 하는데, BasePowerRefactoringAction 클래스에서 리팩토링에 필요한 사전 검사를 수행하고 난 후에 최종적으로 핸들러를 호출해서 리팩토링을 수행하기 때문이다. 기본적인 개발 구조는 IntelliJ에 있는 오픈소스의 클래스를 일부 수정하여 개발하였기 때문에 기본 구조가 유사하며 일부 클래스는 수정하지 않고 사용하였으나 IntelliJ 라이선스 계약을 준수하였다.

Ⅳ. 성능 평가

본 연구를 통해 구현한 에너지 리팩토링 플러그인을 이용하여 리팩토링을 수행하기 전과 후의 두 예제 애플리케이션을 실행할 때 소모되는 전력량을 측정 및 비교 분석함으로써, 본 에너지 리팩토링 플러그인이 실제 안드로이드 모바일 애플리케이션의 실행 시 전력 효율을 높이는 데 효과가 있음을 보였다.

4.1 실험 환경

실험은 안드로이드 버전이 6.0.1이고 배터리 용량이 2550mAh인 삼성 Galaxy S6 스마트폰과 안드로이드 스튜디오를 이용하여 만든 예제 애플리케이션을 기반으로 진행한다. 예제 애플리케이션은 입력한 시간 및 실행 단위에 따라 음원 파일을 반복 재생하는 간단한 기능을 갖추도록 한다.

예제 애플리케이션을 실행하는 동안의 소모 전력의 측정에는 배터리 히스토리안[12]을 이용한다. 구글에서 제작한 배터리 사용량 측정 도구인 배터리 히스토리안은 모바일 애플리케이션의 전력 소비를 시각화해주는 기능이 있으며, 배터리가 가득 찬 상태로부터 버그 리포트를 생성하기 직전까지의 전체 전력 소비량을 측정한다.

4.2 실험 방법

리팩토링을 적용하기 전과 후의 두 예제 애플리케이션을 켜고 임의의 음원 파일에 대해 각각 50회, 100회, 150회, 200회 반복 재생하도록 설정하여, 설정한 횟수만큼 음원 파일을 재생하고 재생한 동안의 전력 소모량을 측정하였다. 그림 3은 Encapsulate Field 기법을 적용한 private field를 보여주고 있으며 그림 4에서는 메서드에 적용한 것을 보여주고 있다.

```
public class MainActivity extends AppCompatActivity {

    private int counts = 0;
    private int seconds = 0;
    private int saveNum = 0;
    private int fileNum = 0;
```

그림 3. Encapsulate Field 기법 적용 후의 Private Field
Fig. 3. Private field after applying encapsulate field

```
public int getCounts() {
    return counts;
}

public void setCounts(int counts) {
    this.counts = counts;
}
```

그림 4. Encapsulate Field 기법 적용 후의 메서드들
Fig. 4. Methods after applying encapsulate field

4.3 기대 효과

본 장에서는 전력 효율이 좋은 리팩토링 기법 중 Encapsulate Field 기법을 수행하기 전과 후의 두 예제 애플리케이션을 실행할 때 소모되는 전력량을 측정 및 비교 분석함으로써, 본 리팩토링 플러그인이 실제 모바일 애플리케이션의 전력 효율을 높이는 데 도움이 될 수 있음을 보였다.

각각의 전력 측정치를 횟수로 나누어 평균을 낸 뒤 비교하였을 때, Encapsulate Field 기법이 적용된 애플리케이션은 적용되기 전의 애플리케이션에 비해 약 4.7%의 전력 소모량이 감소함(표 4)을 확인하였다.

표 4. 리팩토링 전·후 전력 소모량 측정 실험 결과
Table 4. Experiment result of power consumption before and after refactoring

Equipment	Power Usage(mAh)				
App Execution	Number of Repeat Play Times				Average per time
	50	100	150	200	
Before Refactoring	0.134	0.220	0.302	0.424	0.00216
After Refactoring	0.105	0.213	0.302	0.410	0.00206

V. 결론 및 향후 과제

본 논문에서는 모바일 애플리케이션의 전력 소모량을 줄이는 방안으로 안드로이드 애플리케이션 개발 시 안드로이드 스튜디오에 설치하여 사용하는 에너지 리팩토링 플러그인을 설계 및 구현한 내용을 기술하였고, 실제로 예제 애플리케이션에 플러그인을 적용하여 전력 효율이 높아짐을 보였다.

본 논문에서는 이전에 시도된 바 없던, 전력 효율을 위해 소스 코드 단위로 프로그램을 개선하는 시스템을 제안 및 구현하고 실제 모바일 애플리케이션의 전력 소모량을 감소시킬 수 있음을 입증하였다는 점에서 연구에 의의를 찾을 수 있다. 그러나 본 연구에서 개발한 플러그인에 탑재된 에너지 리팩토링 기법들은 근원적으로 소프트웨어 유지 보수를 목적으로 하는 M. Fowler의 리팩토링 기법 중 전력 효율이 좋은 기법을 찾아 적용한 것이며 전력

소모 절감을 목적으로 설계된 리팩토링 기법이 아니라는 점에서 본 연구의 한계를 찾을 수 있다. 따라서 향후 이들 한계점을 극복하기 위하여 추가적인 리팩토링 기법에 관한 연구가 필요할 것이다.

References

[1] H. Li, X. Liu, and Q. Mei, "Predicting Smartphone Battery Life based on Comprehensive and Real-time Usage Data", arXiv:1801.04069, Jan. 2018.

[2] R. W. Ahmad, R. S. Bashir, S. Saeed, Y. Lee, K. Ko, and Y. Son, "Online Cloud-Based Battery Lifetime Estimation Framework for Smartphone Devices", Procedia Computer Science, Vol 110, pp. 70-77, 2017.

[3] O. Choi and S. Chong, "Battery-Aware Data Transmission for Delay-Tolerant Smartphone Applications", Journal of Korean Institute of Communications and Information Sciences, Vol. 41, No. 9, pp. 1054-1056, Sep. 2016.

[4] T. Murakami, S. Kurihara, and S. Fukuda, "Saving Power Consumption of Smartphones in the Screen-off State with Disabling the Wi-Fi", 2018 IEEE International Conference on Consumer Electronics, pp. 1-6, Jan. 2018.

[5] M. Gottschalk, M. Josefiok, J. Jelschen, and A. Winter, "Removing Energy Code Smells with Reengineering Services", GI-Jahrestagung 208, pp. 441-455, 2012.

[6] L. Cruz, R. Abreu, and J. Rouvignac, "Leafactor: Improving Energy Efficiency of Android Apps via Automatic Refactoring", 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, pp. 205-206, 2017.

[7] L. Cruz and R. Abreu, "Using Automatic Refactoring to Improve Energy Efficiency of Android Apps", In Proceedings of the CIbSE XXI Ibero-American Conference on Software Engineering, 2018.

[8] J. J. Park, D. H. Kim, and J. E. Hong, "Analysis of Energy Efficiency for Code Refactoring Techniques", KIPS Transactions on Software and Data Engineering, Vol. 3, No. 3, pp. 109-118, 2014.

[9] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code", Addison Wesley, 2002.

[10] Z. Herczeg, D. Schmidt, A. Kiss, N. Wehn, and T. Gyimothy, "Energy Simulation of Embedded Xscale Systems with XEEMU", Journal of Embedded Computing, pp. 209-219, Aug. 2009.

[11] https://www.jetbrains.org/intellij/sdk/docs/ [accessed: Oct. 30, 2018]

[12] https://github.com/google/battery-historian/ [accessed: Oct. 30, 2018]

저자소개

김 성 보 (Seong-Bo Kim)

2017년 2월 : 성균관대학교 컴퓨터공학과(공학사)
2017년 ~ 현재 : 성균관대학교 전자전기컴퓨터공학과 석사과정
관심분야 : Cloud Computing, Big Data, IoT

최 기 현 (Kee-Hyun Choi)

2000년 2월 : 성균관대학교 전기전자컴퓨터공학부(공학사)
2002년 2월 : 성균관대학교 전기전자컴퓨터공학과(공학석사)
2006년 8월 : 성균관대학교 정보통신공학부(공학박사)
2006년 7월 ~ 2009년 5월: 성균관대학교 BK21 연구교수
2009년 9월 ~ 2012년 8월 : 성균관대학교 학술연구교수
2016년 11월 ~ 현재 : 성균관대학교 리서치펠로우
관심분야 : 무선랜, Active Queue Management, TCP/IP 혼잡제어, MAC 스케줄링

구 자 환 (Ja-Hwan Koo)

1995년 : 성균관대학교 정보공학과 (공학사)
1997년 : 성균관대학교 전기전자 컴퓨터공학과 (공학석사)
1999년 ~ 2002년 : LG CNS 정보기술연구소 연구원
2006년 : 성균관대학교 전기전자컴퓨터공학과(공학박사)
2007년 ~ 2010년 : 미국 위스콘신대학교 컴퓨터과학과 박사후 연구원
2016년 ~ 현재 : 성균관대학교 사회과학대학 연구교수
관심분야 : Data Communication, Cloud Computing, Big Data

김 응 모 (Ung-Mo Kim)

1981년 : 성균관대학교 수학과 (학사)
1986년 : 미국 Old Dominion 대학교 컴퓨터과학과 (공학석사)
1990년 : 미국 Northwestern 대학교 컴퓨터과학과 (공학박사)
1990년 ~ 현재 : 성균관대학교 소프트웨어대학 교수
관심분야 : Database, Data Mining, Big Data