A Reproduction of Dependency Facade: The Coupling and Conflicts between Android Framework and Its Customization

This package includes the artifact associated with the article title "Dependency Facade: The Coupling and Conflicts between Android Framework and Its Customization". The content of the package includes the tool, scripts and experimental data used in the study.

❖ Overview

Section I: README

I-I: Content of this artifact package

The artifact folder we shared is named ***DepFCD_AE***, it includes the artifacts that correspond to results presented in our paper, and it also includes reusable tools/scripts for similar future studies that we intent to share publicly.

The ***DepFCD_AE*** folder is located online: https://github.com/xjtu-enre/DepFCD

Once the ***DepFCD_AE*.zip** file is downloaded, unpack it. Then the following structure should be seen:

- Method (DepFCD tool)
  - enre_java.jar
  - dep_facade.exe

  - ref_tool
- Scripts
  - Setup_scripts
  - RQ_scripts
- Data
  - Methodology
  - Setup
  - Results
- Document
  - README.docx
  - INSTALL.docx
  - LICENSE
  - REQUIREMENTS.docxS
  - STATUS.docx
- icse2023main-p2167.pdf (the research paper submitted and accepted)

The **method, scrpits** and **data** folders include the analysis tool and our data, respectively. Due to their large sizes, we could not include (1) all the json files, nor (2) git log for all versions of projects. Also, since our entire study took almost a year to finish, for this artifact evaluation, it is not feasible to reproduce all the original traces (for entity ownership identification) either, nor is it feasible to reproduce all the installation logs (for the merge points and conflicts reproduction). However, to facilitate this artifact evaluation and show the reusability of our utilities, we provided all the experimentation scripts, and gave instructions on how to use them against simple samples (see the 'Partial reproduction' section below).

I-II: Explore the artifact

First, clone the source code (i.e. `/framework/base` of LineageOS), then explore as follows

Most of the following steps concern only browsing original inputs to the study, intermediate files produced, raw study results, and final results reported in research paper. We provide complete instructions to reproduce the study and some intermediate results to facilitate reproduction due to the total time cost of the study. Our study needed to clone 5 open-source projects on 13 different versions(each for a different Android version); more critically, it needed to analyze 13 'framework/base' and run each individual code for 10 minutes by enre-java.jar and nearly 1 days by dep_facade.exe, then repeat this process for each projects. The trace storage space was over 100GB.

1. Check the study tools (the 'Method' folder)

For the interface-level dependencies, we employ the **enre_java.jar** (https://github.com/xjtu-enre/ENRE-java) to extract the entities and dependencies from the source code. For the intrusion-level dependencies, we provide **dep_facade.exe** -- a dependency facade analysis tool to detect entities' ownership and intrusive operation in the

dependency graph. In this tool, we integrated the tool RefactoringMiner under **/ref_tool/lib** (https://github.com/tsantalis/RefactoringMiner)

2. Check the analysis scripts (the 'Scripts' folder)

For the merge points and conflicts, we use the following scripts to get merge points and conflicts. You can refer to https://zenodo.org/record/6272071#.Yxg_OWhBw_E for more information.

For the analysis results of RQ3 and RQ4, we use the following scripts to integrate the data we obtained above.

3. Check the study data (the 'Data' folder)

The following list provides a mapping between the generated results and parts in the research paper, along with the note on how the figure/table was produced.

- **Methodology (The detection of Dependency facade)**
  - Entity and Dependency Extraction
    - *<project name>-<version>*-out.json - which contains the corresponding dependency graph.
  - Entity Ownership Identification
    - all_base_commits.csv - the evolution history of all commits of the upstream AOSP frameworks/base
    - *<project name>-<version>*
      - blame_dict.csv - git blame detection result of the current project selected merge node
      - accompany_commits.csv - information of current project version's commit history
      - base_commits.csv - information of merge point which downstream project version merging upstream AOSP
      - old_base_commits.csv - commit history of current project version which belongs to upstream AOSP old version
      - only_accompany_commits.csv - commit history of current project version which belongs to downstream project
      - all_entities.csv - all entities and commits that contributed to specific entity information of files that contains an intrusive modification of the downstream project
      - final_ownership.csv - the entity ownership detection result below the 'File' level in the project
      - facade.json - the detection result of dependency facade of current project version
  - Intrusive Operation Identification
    - *<project name>-<version>*
      - access_modify_entities.csv - The number of entities modified by accessibility
      - final_modify_entities.csv - The number of entities modified by 'final'
      - annotation_modify_entities.csv

- add_import.json - The num of newly-added statements of 'import'
- inner_extensive_class_entities.csv - The num of newly added inner classes in the native class
- parent_class_modify_entities.csv - The number of classes modified by the parent class('inherit')
- parent_interface_modify_entities.csv - The number of classes modified by the parent interface('interface')
- class_var_extensive_entities.csv - The number of newly added class field
- class_var_modify_entities.csv - The number of modified class field
- method_var_modify_entities.csv - The number of modified variables in the method
- param_modify_entities.csv - The number of methods modified by parameters
- return_type_modify_entities.csv - The number of methods modified by return type
- refactor_entities.csv - The number of refactoring entities (rename, extract, and others)

○ Restriction Level Labeling

- matching.xlsx - the matching statistics of non-SDK restriction files to the dependency graph which is provided by ENRE
- *hiddenapi-flags-<project name>-<version>*.csv - the 'hiddenapi-flags.csv' of downstream LineageOS 18.1 and 19.1, upstream AOSP 11 and 12.

- **Set up**

○ Subject and Version Collection (TABLE I)

- *<project name>-<version>-merge.csv* - the merge points of the current downstream project version which contains merge commit, both upstream and downstream's parent commits and branches.

○ Merge Conflict Collection (TABLE I)

- *<project name>-<version>-merge.csv* - the conflict details of the current downstream project version which contains merge nodes, conflict files quantity, conflict java files quantity, and conflict blocks LOC.

- **Results**

○ RQ1: How do the downstream customizations rely on the upstream Android through interface-level dependencies?

- entity ownership distribution (TABLE III)
  - entity-ownership.xlsx - the final collection of all project versions' entity ownership detection results
  - <project name>-<version>
    - final_ownership_count.csv - the number of each kind of entity combined with the ownership detection result of the current project version
- facade size measurements (Fig. 4)
  - facade.xlsx - the collection of all downstream project versions' dependency facade detection results
  - <project name>-<version>
    - facade_base_info_count.csv - The number of entities and dependencies appearing in the dependency facade

        - facade_relation_info_count.csv - The number of each kind of dependency appearing in the dependency facade
        - facade_file_filter.csv - The number of each kind of entity appearing in the dependency facade which is counted in files
    - interface-level dependencies (D->U) (TABLE II)
      - interface-level dep.xlsx - the collection of all downstream project versions' interface-level dependencies facade detection results
      - <project name>-<version>
        - interface_level_facade_d2u.csv - The number of each kind of dependency appearing in the dependency facade(D->U)
- RQ2: How do the downstream customizations rely on the upstream Android through intrusion-level dependencies?
    - intrusive operations (TABLE IV)
      - intrusive-type.xlsx - The collection of different kinds of intrusive modifications quantity for all project version
      - <project name>-<version>
        - intrusive_count.csv - The quantity of each kind of intrusive modification
        - intrusive_file_count.csv - The quantity of each kind of intrusive modification which is counted in files
    - reverse dependencies (TABLE V)
      - <project name>-<version>
        - interface_level_facade_u2d.csv - The number of each kind of dependency appearing in the dependency facade(U->D)
- RQ3: How do the downstream customizations adapt to the dependency constraint imposed by the upstream Android?
    - facade_hidden_filter_lineage_18.1.csv - the usage of hidden on the dependency facade of lineage 18.1 (TABLE VI)
    - facade_hidden_filter_lineage_19.1.csv - the usage of hidden on the dependency facade of lineage 19.1 (TABLE VI)
    - hidden-usage.xlsx - the summary of usage of hidden for both LineageOS and IndustrialX (TABLE VI)
    - hiddenapi-flags-change.xlsx - the change of hidden flags during version evolution. (Fig. 7 and Fig. 8)
    - hiddenapi-matching.xlsx - the matching between hidden flags and dependency graph (THREATS TO VALIDIT)
- RQ4: How do merge conflicts occur on the dependency facade between downstream customizations and the upstream Android?
    - selected-conf-block.docx - the selected conflict blocks details. (TABLE VIII)

Section II: REQUIREMENTSS

Due to the large size of the AOSP and its customizations, the memory heap of the machine is better to over 30G. Our artifacts have been tested with the following environment settings and tools/libraries (i.e., the configurations on a clean machine):

- Java 11 or higher
- Python 3.7(+)
- Python3 packages: Gitpython
- latest version of Git
- latest version of pip3
- All the libraries for dep_facade.exe as listed in Method\ref_tool**lib**

Section III: INSTALL(Reproduce the artifact)

As explained above, fully reproducing all the results presented in the research paper would not be feasible with respect to the time for this artifact evaluation. However, to validate that the study tools and data actually worked to enable our full study and to show the reusability of our utilities, we prepared a simply way for reproduction.

A very basic usage example for reproduction means: reproducing our study on one version of projects (i.e. LineageOS-18.1) but covering the complete study process (i.e., from intermediate data to final results). We provide detailed instructions to prepare an executable environment, install the tool and how to test it with example.

1. Prepare executable environment

1) JDK environment:
Please download 11(+) JDK package at the official site :
https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html
Then follow the official installation guide to set PATH:
https://docs.oracle.com/en/java/javase/19/install/overview-jdk-installation.html#GUID-8677A77F-231A-40F7-98B9-1FD0B48C346A
2) Python environment:
Please download 3.7(+) python at the official site:
https://www.python.org/downloads/
3) Upgrade pip to latest version
Run command at console: pip3 install --upgrade pip
4) Install Git
Please follow the official guide to install Git:
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
5) Install Gitpython:
Run command at console: pip3 install Gitpython

2. Install DepFCD

DepFCD is an executable file which is located under the 'Method' folder after unpacking our artifact. The detailed instructions to use it are provided below.

3. Prepare test data

1) Clone corresponding customized Android Frameworks into directory platforms.
To clone Android /framework/base, do:
git clone https://android.googlesource.com/platform/frameworks/base
To clone LineageOS /framework/base, do:
git clone https://github.com/LineageOS/android_frameworks_base.git

2) Reset the version of the downstream and the corresponding upstream Android projects.
For Example, for LineagsOS-18.1, the merge point is:

Merge point	LineageOS branch	LineageOS parent commit	Android branch	Android parent commit
7f7fc2562a95be630dbe609e8fb70383dcfada4f	lineage-18.1	cf0606c2f73	android11-gsi	49d8b986ddd

To reset the code status for Android, do:
git checkout android11-gsi
git reset --hard 49d8b986ddd
To reset the code status for LineageOS, do:
git checkout lineage-18.1
git reset --hard cf0606c2f73

4. Entity and Dependency Extraction

To execute `enre_java.jar`, we need to specify source code path, hidden flag file path, do

java -jar enre_java.jar java <local_path_to_repo\LineageOS\base> lineage -hd hiddenapi-flags.csv -o <output_file_name>

5. Entity Ownership and Intrusive Operation Identification (DepFCD)

We integrate the tool `RefactoringMiner` including

- `ref_tool\lib\RefactoringMiner-2.2.0.jar` - the refactoring infomation detection tool, there are some `jar` required for the tool in the `ref_tool\lib` directory
- `ref_tool\bin\RefactoringMiner` - a script that executes the tool `Refactoring Miner-2.2.0.jar` to generate refactoring information

To reproduce, we need to specify source code path, source dependencies graph json file path, do

dep_facade.exe -cc <local_path_to_downstream_repo\LineageOS\base> -ca <local_path_to_upstream_repo\android\base> -c <path_to_downstream_dependency_graph, i.e. D:\LineageOS\lineage.json> -a <path_to_upstream_dependency_graph, i.e. D:\android\android.json> -ref ref_tool\bin\RefactoringMiner -o <output_path>

During the detecting process，we need to acquire all commits history and retrieve the whole refactoring operation of the analyzed project, which is time-consuming. To be convenient, you can use the `all_base_commits.csv` and `ref.json` for LineageOS-18.1 which is the result of RefactoringMiner in the folder `data\Methodology\The detection of Dependency facade \Entity Ownership Identification`. The only thing we need to do is moving them to the `output_path` which is specified in the above command.

The WARNING related to `log4j` during execution can be ignorable.

Section Ⅳ: STATUS(badges the authors are applying for)

We are claiming 3 badges (Functional, Reproduced, and Available) under the Artifacts Evaluated and Available sections.

1. Functional. The artifact is completely documented to record an inventory of the tool, the scripts, the results, and the documentation about how to exercise the artifact. The artifact is consistent because it contains DepFCD and the intermediate data used to produce the results in the technical paper. The artifact is complete because it includes all the components relevant to the technical paper. The artifact is exercisable because the scripts to produce the results can execute successfully using the commands provided in the documentation.

2. Reproduced. We strongly believe that our DepFCD tool can be reproduced and extended. DepFCD provides a user-friendly command-line interface, prompting help and usage guidance. The source code of DepFCD is written in Java and Python, hence easy to extend its functionality. Moreover, the dataset we provided is clean and complete to facilitate continuous research in the community.

3. Available. We publish the whole data, scripts and tools which also includes processed data of close-source IndustrialX at https://github.com/DepFCD/DepFCD so that the community can reuse, improve, and extend it.