Development: Android Architecture Visualizer

Analyzing APK Files to Recreate Software Architecture Diagram using COVERT and ArchStudio

Tim Roty
Software Engineering
University of Nebraska-Lincoln
tim.roty@huskers.unl.edu

Rachel Nordgren
Software Engineering
University of Nebraska-Lincoln
Rachel.nordgren@huskers.unl.edu

Ian Anderson
Software Engineering
University of Nebraska-Lincoln
Ian.Anderson@huskers.unl.edu

Austin Collins
Software Engineering
University of Nebraska-Lincoln
Collins.Austin@huskers.unl.edu

Laura Derowitsch
Software Engineering
University of Nebraska-Lincoln
Derowitsch.Laura@huskers.unl.edu

ABSTRACT

Our project is the middle piece in a pipeline to analyze an Android application's architecture. We utilize COVERT software to break down an APK file, use that broken-down file to develop an XML file, and input the XML file created into ArchStudio in Eclipse to see the visual software architecture diagram. Our project helps software teams ensure the designed software architecture matches the architecture of the production software. This can help minimize costs related to incorrect architecture. The project also helps to ensure that software architecture is something that is considered throughout the entire lifecycle of a software application and not just a first step in the design process. Our report will break down the background and motivation for our project, our methodology, how to run and test our project on our provided sample data, an overview of what is included in the deliverables, and the potential future of our work.

INTRODUCTION

A critical component to a project's success throughout the software life cycle is a disciplined approach to software architecture. Selecting and implementing a consistent architecture maximizes team velocity and aids in the division of work throughout the process of development. Architecture documentation helps to structure a project to meet business requirements and enforce quality standards. However, an architecture plan rarely makes it all the way through to production. One enemy of a consistent architecture is architecture drift, or erosion, where software systems implement an unintended and undocumented architecture. To overcome this enemy, software systems must undergo architecture recovery, which is the process of "inspecting software systems and extracting representations of their artefacts for the purpose of documenting the software architecture." (Tamburri & Kazman, 2017)

When architecture drift occurs, the intended architecture is not broken, but decisions exist that were not accounted for. Because the system is not broken, the drift often goes ignored and unresolved by engineers, since it does not seem to be a relevant concern at the time. However, drifting architecture threatens the success of a project because the quality guaranteed by the intended architecture erodes. Our tool helps engineers prevent architecture drift and aids them in the process of architecture recovery through a proactive approach. This process allows engineers to compare the software architecture of their product to the intended design to identify any diversion in their structure. It is often difficult and expensive to create architecture diagrams every time a project's architecture changes. However, our tool allows easy conversion from a built application to an architecture diagram. Engineers can use the tool at any point in the development process to ensure they are following the intended architecture without additional work. This alleviates high costs of software recovery, since it is a proactive approach and eliminates costs associated with engineers documenting architecture to create other diagrams rather than developing. Every software decision and update can have a documented software architecture diagram with little to no effort.

If software developers consistently use our tool, large erosion costs can be avoided by tech companies, and keeping consistency as systems become more complex and engineers come and go can become much easier. If architecture changes along the way are documented via diagram, it makes it easier for new developers to pick up the project and continue the intended architecture. It also ensures the intended purpose and design of the system will remain the same, which improves the overall quality of the system.

BACKGROUND

Software architecture is arguably the most critical part of the design and development process. Yet oftentimes, architecture is discussed and potentially diagrammed early on in the development life cycle, but this attention to architecture is not followed through during the rest of the life cycle. To see how our tool can help development teams to improve their architecture throughout the development process, it is important to understand the terminology, tools, and methods used in this project.

The core of the problem we attempt to solve is the common discrepancy between prescriptive and descriptive architecture. Prescriptive architecture is the designed architecture before implementation whereas descriptive architecture is the architecture of the software after implementation. It is rare for perspective and descriptive architecture to match due to the many changes that are likely to occur during the development process. As new features are added to the software the descriptive architecture gets further and further from the prescriptive architecture. This is called architecture drift. Architecture drift is an expected part of software development, but too much can become a serious problem. It is important to keep up documentation with the descriptive state of your architecture so that other developers, QAs, and testers can easily understand the software.

Architecture is costly to fix, and it only becomes more expensive with more architecture drift. Therefore, consistent audits of software architecture can save money in the long run as making small architecture changes and updates as you go is much easier than making huge changes after a long time of architecture drift.

There are also a few other terms that will be used throughout this project that some may not be familiar with. The first is APK. An APK is an Android application package. It is essentially the file for a mobile application that runs on the Android operating system. The other file format we talk about is XML, which stands for eXtensible Markup Language. This is the file format that our program outputs, which can then be visualized in other software. It stores the data behind the architecture diagram.

The other piece of software that the project relies on is a program called COVERT: Compositional Analysis of Android Inter-Application Vulnerabilities [1]. “COVERT is a tool for compositional verification of Android inter-application vulnerabilities,” but it has a feature that breaks down an APK into an XML file detailing the components and intents within the app. This gives us the information necessary to create an XML file that visualizes the software architecture.

There was another piece of software we considered using instead of COVERT: Androguard [2]. It is a similar software, but there were a few reasons that we chose to use COVERT over Androguard. The primary one was that the output of COVERT was much more easily convertible into XML architecture diagrams. COVERT, unlike Androguard, gave an output that included all components and intents in the APK in a way that is easy to parse. Additionally, Androguard was more complex of a software than we needed for our purposes. Since it is required to have a software like COVERT or Androguard packaged with our software to be able to run, it made more sense to use the software that wasn’t overly complex for our purpose.

The last software that our project relies on is ArchStudio, a system architecture development environment that is an extension for the Eclipse IDE [3]. This is the software that is used to visualize the XML architecture diagrams.

APPROACH

The goal of our project was to enable the architecture of a working android APK file to be visually analyzed. To achieve this goal, we used COVERT to analyze the APK, converted that into a usable JSON object, and then converted the output into the format required by Archstudio. Figure 1 below shows a high-level overview of the project workflow.

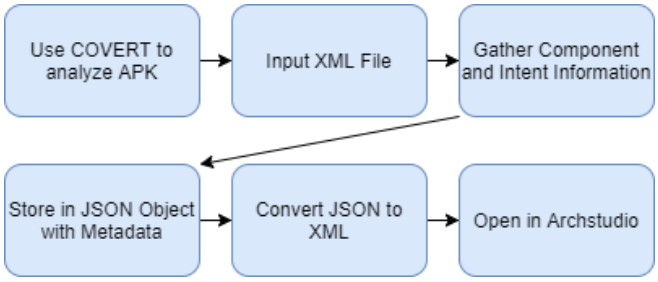

Figure 1: Flowchart displaying the project workflow/ pipeline of the final deliverable

The first step was to break the APK down into a formal model of the application. We used COVERT, an existing android analysis tool, to obtain an XML file containing details of all components, intents, and other aspects of the APK. We included COVERT and sample output in the project deliverables for convenience for testers. Figure 2 shows an example component in the format output by COVERT.

```
<components>
    <Component>
        <type>activity</type>
        <name>com.sdalab.messenger.MainActivity</name>
        <RequiredPermissions/>
        <IntentFilter>
            <filter>
                <actions>android.intent.action.MAIN</actions>
                <categories>android.intent.category.LAUNCHER</categories>
                <pathData></pathData>
            </filter>
        </IntentFilter>
        <PropagatedPermissions/>
    </Component>
```

Figure 2: An XML file that contains information about a component in an Android APK from the COVERT output model.

Before this XML output can be visually analyzed by a software architect, it needs to be converted into a format that can be opened in ArchStudio. To achieve this, the model is opened in our software which breaks it down into components and their respective intents. For each component in the APK, we store the name and type of the component as well as a pair of X and Y coordinates, created dynamically based on the number of components, to create a grid on the Archstudio diagram. We then loop through all the intents in the APK and create a link between the two components that the intent interacts with. The link also stores which component was the sender and which was the receiver to be used when creating the in and out interfaces in Archstudio. These components and links are all added to one JSON object containing everything needed by Archstudio. Figure 3 shows a screenshot of the code that builds the component portion of the JSON objects.

```
let componentNum = 1;
jsonComponents.forEach(element => {

    let name = element["elements"][1]["elements"][0]["text"]
    let type = element["elements"][0]["elements"][0]["text"]
    let id = "componentId" + componentNum;

    let x = ((componentNum * WIDTH) + (componentNum * OFFSET)) % ((WIDTH + OFFSET) * NUM_WIDE);
    let y = (HEIGHT + OFFSET) * Math.floor(componentNum / NUM_WIDE);

    var component = {
        "name": name,
        "type": type,
        "id": id,
        "x_coord": x,
        "y_coord": y,
        "height": HEIGHT,
        "width": WIDTH
    }

    middleObject.components[name] = component
    middleObject.components[name].interface = [];

    componentNum++;
});
```

Figure 3: Code depicting the organizing of a COVERT model's components into JSON components that will be used to build the final program output.

The final process is converting this JSON object back into XML that can be opened in Archstudio. We add the metadata required by an Archstudio model and then each of the components and links that we gathered from the previous portion. This is all formatted following ArchStudio's conventions and converted to an XML output file using preexisting NPM packages. This XML output file can then be opened in Archstudio to visually analyze the architecture of the given Android application.

This tool achieves our goal of allowing architects to quickly analyze an Android APK. Using this tool users can take any APK, run it through the process described above and easily see an architecture model of the current state of the application. This is useful when validating that the descriptive architecture aligns with the prescriptive architecture.

EVALUATION

To evaluate the success of our project, we recommend using the test data provided and following the steps below to test out our software. The first step is installing dependencies. The only dependencies we have are NPM packages, which are installed using **npm install** in the main folder of deliverables. Once these are installed, we can continue with the project. As described in the Approach section, there are three main steps in our pipeline, running COVERT, running our application, and running ArchStudio.

To run COVERT, you must first install COVERT, which is compatible with Java 8 or older. Java 9 and newer are not

compatible with COVERT, so it is important to check your version of Java before trying to run the application. We have provided two sample APK files to input into COVERT for testing purposes. These can be found in the "Sample Android APK Data" folder. After selecting which sample data to use, place all APK files to be analyzed in the demo folder, located at covert > app_repo > demo, which will allow for COVERT to run the analysis.

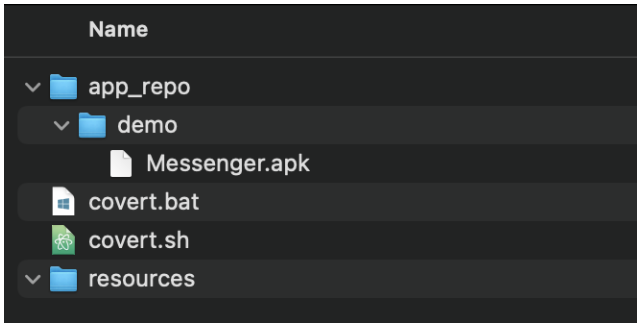

Figure 4: Depicts the file structure needed to run the COVERT application. The Messenger.apk file is replaced with the APK file that will be analyzed.

COVERT is run from the command line. In the command line, navigate inside the COVERT folder from your file system. Next, to start the application on the demo, you need to run **./covert.sh demo** , which will start steps that analyze the inputted APK file. If you wish to change which file you are analyzing, simply move the files in and out of the demo folder and run the command again. More information can be found on the COVERT documentation website, referenced below.

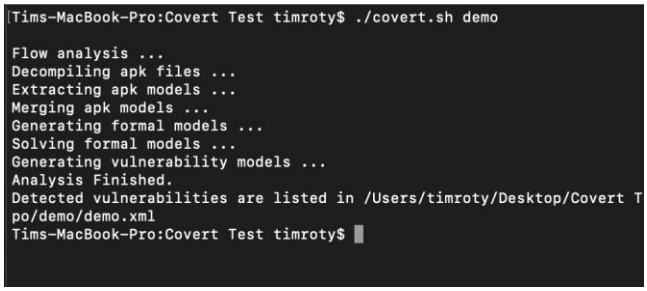

Figure 5: Example command and output from the COVERT program analyzing an Android APK file.

Now that the APK file has been analyzed with COVERT, our application will be used. Our application runs on the UNL CSE server on its current version of Node, v8.17.0, and the current version of NPM, 6.13.4. Our application is also run from the command line and requires a single argument, which is the file path for the analyzed APK file from COVERT. Our team copied and pasted those files into the main folder to streamline the process and create an easier file path specification for testing purposes. While that is recommended, that is not necessary. If the file path is invalid or the path leads to a file of an incorrect file type, the program will error out and provide improper results.

To run our application, use the command node **./application.js <filePath>** , where this assumes the model has been copied to the same location as the application. When the application is complete, it will give a prompt confirming the successful output and create a file called "ArchStudioXML.xml" in the same folder as the application. This output will be the input for our ArchStudio project.

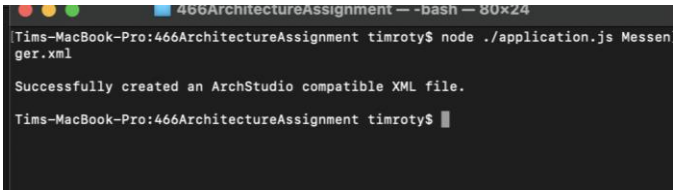

Figure 6: Example command and output from the created program used to create the ArchStudio compatible file.

Our team runs ArchStudio on Java version '1.8.0_171', but ArchStudio is more flexible with Java versions. We also run it on Eclipse 4.8. To run the project on ArchStudio, create a new project in Eclipse and drag the "ArchStudioXML.xml" file from our application into the project. From there, right click the newly added file and open it using Archipelago v2.0. Click the created program structure and see the Android APK file's architecture.

This process is how we ensure our entire project pipeline runs correctly. For more detailed instruction, see our project's README attached to the repository. This process allows us to check the accuracy and usability of our software. We've included two different test files to run the process on for your convenience.

CONCLUSION AND FUTURE WORK

The main highlights of our project are our process and our purpose. The process of our project utilizes COVERT, a pre-existing tool that can be leveraged by developer teams, to set up our project for success. Using COVERT's output, we created a process to extract the software architecture of an Android application. Our process is streamlined and convenient for developers to use multiple times throughout the life cycle of a software project. The purpose of our

project is another highlight. We spent a great deal of time in class discussing the importance of software architecture, not just as an initial step of the development process, but as a major factor that needs to be considered throughout the development process. If utilized correctly and frequently, our project could help a software team minimize costs and time spent redoing software when the architecture goes off plan.

Our team feels that our project was successful for the scope of the assignment, but we also see the future of this project as much bigger. Software architecture is always important to a software team and will only continue to be more important as software applications grow and scale. We can see our project becoming a very useful tool for lots of companies. The biggest future of our project would be getting COVERT and ArchStudio to run within a greater, single application. While we currently use it as a third-party tool, it would be great for a user to simply upload an APK file and get the architecture diagram out with one step. While this lofty goal was out of our semester scope, this would encourage developers to use the tool more frequently as it would take a lot less effort from the user. Another goal for the project would be to get interfaces to display more spaced out and to potentially be able to compare designed architecture with the actual architecture model to directly show the user the discrepancies between the two architectural models. Between the streamlined process and the comparison between two architectures, our project has a lot of merit for the future going forward to make our contributions meaningful for the greater software community.

DELIVERABLES

Our deliverables that are attached with this report are our source code, data for experiential tests, models of our project, COVERT software to run on the CSE server. We have included a README documentation with our source code to further describe how to use our project, as well as COVERT and ArchStudio. The README describes which versions of the software applications to use, commands to run in the command line to run COVERT and our application, as well as how to install the necessary dependencies.

DIVISION OF WORK

Our entire team had a very hands-on approach for the project, so everyone contributed to each step of the project. We met up and wrote the project proposal together, brainstorming what projects we wanted to do and split up initial research to ensure our decided project was within our means. Once we decided to move forward with our idea, we met as a team to develop the architecture and pair program the project. We also split up work for the project presentation, this report, and final deliverables. Here is a break down on the work each team member had the highest contributions for:

Tim: project research, architecture design, project development, report

Laura: project research, project development, presentation, report

Rachel: project research, project development, presentation, report

Austin: project research, project development, report

ACKNOWLEDGMENTS

The team would like to thank Professor Hamid Bagheri and Teaching Assistant Jianghao Wang for their help in assisting and guiding the team through the project ideation and development.

REFERENCES

[1] H. Bagheri, “Lecture 4 - Basic Concepts,” in CSCE - 466: Software Design and Architecture.

[2] "SEAL - COVERT", Seal.ics.uci.edu, 2021. [Online]. Available: https://seal.ics.uci.edu/projects/covert/. [Accessed: 28- Apr- 2021]. [3] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI:https://doi.org/10.1007/3-540-09237-4.

[4] Anthony Desnos. 2016. Androguard. Retrieved March 12th, 2021 from https://github.com/androguard/androguard

[4] University of California Irvine. 2015. Retrieved March 10th, 2021 from http://isr.uci.edu/projects/archstudio/index.html

[5] D. Tamburri and R. Kazman, "General methods for software architecture recovery: a potential approach and its evaluation", Springer Link, 2021.

[6] S. Bachmann, A. Desnos and G. Gueguen, "Welcome to Androguard’s documentation! — Androguard 3.4.0 documentation", Androguard.readthedocs.io, 2021. [Online]. Available:

https://androguard.readthedocs.io/en/latest/. [Accessed: 28- Apr- 2021].