Mestrado em Engenharia Informática

Dissertação/Estágio - Relatório final 2017/2018

Source code analysis and transformation to aid internationalization support

João Pedro Santos Batanete

joaosb@student.dei.uc.pt

July 2018

Orientador DEI:

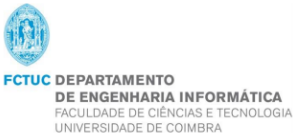
Ernesto Costa

Orientador WIT:

João Certo

Tutor WIT:

Carlos Mota



Acknowledgements

The present document marks the end of my tenure as Software Engineering student at the University of Coimbra.

Firstly, I would like to thank both my supervisors and my tutor for all the feedback provided and all those times you pushed me in the right direction when I needed it. I have learned a lot from you, and I will continue to do my best towards improving myself as a professional in this field. I would also like to thank all the people at WIT-Software, who were always very friendly and welcoming to me, and provided invaluable feedback when I needed it.

I would also like to thank my parents, sister and the rest of my family for all the support they gave me across all these years.

Lastly, I would also like to thank my girlfriend for always cheering me up when I needed it the most and inspiring me to always try my best.

Abstract

With the widespread expansion of software markets, localization is becoming increasingly important. There is a pronounced correlation between application usage and availability in the home language. This creates the need to improve localization support processes in order to deliver applications more quickly and cost-effectively.

In order to manage textual content, one commonly used approach is the use of key/value pairs to represent user interface text fragments with the values for each key, and for each target idiom being stored in a localization platform, which is used by the translating team to create the translations.

Translators often need ”contextual” information when translating a given key in order to provide an acceptable translation for it. In order to achieve this, it is common to use screenshots of the application context (user interface) associated with each key that appears on-screen. The process of obtaining the associations is currently done manually, which is very time consuming and sometimes impractical for real-world applications where the number of keys is too large or the application’s user interface is too complex.

The central objective of this internship is to create a tool that automatizes certain portions of the process by aiding the user in obtaining associations between the key/value pairs and the screenshots and uploading them to a localization platform. This is achieved by changing application source code directly using static code analysis and transformation tools. The platforms supported initially will be Android and iOS but extensibility to other platforms is a primary goal.

Keywords

”Android”, ”iOS”, ”Static code analysis”, ”Code transformation”, ”Localization”, ”Internationalization”, ”ANTLR”, ”Key/value localization”

Resumo

Com a expansão dos mercados de software, a localização dos produtos está a tornar-se cada vez mais importante. Existe uma correlação pronunciada entre o sucesso das aplicações e a disponibilidade destas para o idioma local. Isto cria a necessidade de melhorar os processos de suporte à localização para possibilitar o desenvolvimento de aplicações de forma mais eficiente em termos de custos e tempo.

Para gerir conteúdo textual, uma abordagem comum é a utilização de pares chave/valor para representar os fragmentos de texto da *user interface*, com os valores correspondentes a cada chave e a cada idioma a serem armazenados numa plataforma de localização, que é utilizada pela equipa de tradução para criar as traduções.

Os tradutores necessitam com frequência de informação "contextual" para durante o processo de tradução de uma chave, de forma a conseguirem disponibilizar uma tradução aceitável. Por forma a cumprir este requisito, é comum utilizar fotografias do contexto da aplicação (*user interface*) associadas com as chaves que aparecem no ecrã do dispositivo. O processo de obtenção das associações é, neste momento, realizado de forma manual, o que o torna moroso e por vezes impraticável para aplicações em contexto real com um número de chaves demasiado grande, ou com uma *user interface* demasiado complexa.

O objetivo central deste estágio é criar uma ferramenta que automatiza certas componentes do processo, auxiliando o utilizador na obtenção das associações entre os pares chave/valor e as fotografias e exportando-as para uma plataforma de localização. Isto é feito através da modificação do código fonte de forma direta utilizando ferramentas de análise e transformação estática de código. As plataformas suportadas inicialmente irão ser Android e iOS, mas a extensibilidade a outras plataformas é um dos objetivos principais.

Palavras-chave

"Android", "iOS", "Análise estática de código", "Transformação de código", "Localização", "Internationacionalização", "ANTLR", "Localização com pares chave/valor"

Contents

List of Figures

List of Tables

Acronyms

AST	Abstract Syntax Tree
ASCII	American Standard Code for Information Interchange
BNF	Backus–Naur form grammar
CFG	Control Flow Graph
DSL	Domain Specific Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
i18n	Software Internationalization
l10n	Software Localization
JSON	JavaScript Object Notation
KVS	Key/Value/Screenshot(s) association
REST	Representational State Transfer
RTL	Right-To-Left
UI	User Interface
XML	eXtensible Markup Language

Glossary

Term	Definition
Abstract Syntax Tree (AST)	A tree representation of the syntactic structure of code written in a programming language with focus on the input language's constructs.
Activity(Android)	An Android class which represents an application context where the user interacts with it (in contrast to Services, which run in the background). [2]
Android Studio	An Integrated Development Environment developed by Google. It supports a number of programming languages, including C/C++, Java, Kotlin and Python.
Backlog (Agile methodology)	A list of features or technical tasks which the team maintains and which, at a given moment, are known to be necessary and sufficient to complete a project or a release. [42]
Category (Objective-C)	A feature of the Objective-C language which allows adding functionality to an existing class (for example, adding new methods or attributes).

CocoaPods	A dependency manager for Swift and Objective-C Cocoa projects implemented in the Xcode IDE. It is widely used in projects for both macOS and iOS. Unlike Gradle, it does not come pre-installed with Xcode and is an open source project unrelated to Apple. [18]
Domain Specific Language (DSL)	Computer or programming language specialized to a particular application domain. [26]
Generative programming	A style of computer programming that uses automated source code creation through generic frames, classes, prototypes, templates, aspects and code generators to improve programmer productivity.
Gradle (Android)	An open-source build automation system and dependency manager that facilitates the build step of application development. Syntax is based on the Groovy programming language.
Integration testing	A phase of software testing where the entire system is tested as a group.
Internationalization (i18n)	The process of designing a software application so that it can be adapted to various languages and regions without engineering changes. [40]
JDOM2	A Java library used to analyze and transform XML files.

Key/Value approach (localization)	An approach to software localization which consists of associating each User Interface text fragment with a given ”key”, and different values for each target language.
Key/Value/Screenshot association(datasets)	An association between a KV pair and one or more screenshots from the UI. In the context of this report, they are referred to as ”datasets”.
Localization (l10n)	The process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text. [40]
l10n platform	A platform that aims to simplify the localization process. It usually consists of a local or Internet hosted web platform where translators and developers collaborate in order to translate a given application.
Machine Translation (MT)	A sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another.
Method Swizzling	A reflection technique used for changing a method's implementation at runtime by changing the reference used to access the method to a different one. [30]
Parse Tree	A tree representation of the syntactic structure of code written in a programming language with focus on its grammar rules.

Pbxproj file	A file extension used by Xcode to represent the structure of a project. Information includes files which are included in the build, compiler flags among others.
Podfile	A file used to manage an Xcode project's CocoaPods dependencies. The syntax is based on the Ruby programming language.
Reflection (programming)	The ability of a computer program to examine, introspect, and/or modify its own structure and behavior at runtime. Different languages possess different reflection features, with some having almost no support for it at all, and others allowing the programmer to freely analyze and manipulate most of its code structures.
Settings bundle (iOS)	A package contained inside iOS projects which allows the application to feature a preferences, accessible from the iPhone device's settings menu.
Translation Memory (localization)	A feature that some localization support tools have, that allows them to save translations for given text segments, in order to use them in future projects.
Token (parsing tools)	A "terminal" symbol of a tree representation structure. They are the elementary units in the syntactic definition of source code, and are produced during lexical analysis. [34]

TokenStreamRewriter	An class from the ANTLR tool used to add text after specific parse tree nodes.
Unit testing	A phase of software testing where each of the system's units of code or modules are tested individually.
View (Android)	The basic building block for UI components in Android applications. UI components such as buttons, textfields and ViewGroups extend from the View class. [6]
ViewGroup (Android)	A special type of view that may contain other views, organized in different ways according to the ViewGroup subclass used. [7]
Xcode	An Integrated Development Environment developed by Apple for macOS and the primary IDE used to building applications for Apple platforms.

1 Introduction

1.1 Motivation

In the last few decades, access to information has become increasingly widespread. Software products are used across a multitude of different countries and cultures. Development also has, over time, adapted.

Software is often developed for a variety of different markets. This process is called software localization and is one of the most important processes in software development today [40].

Localizing an application for any particular market may have a dramatic effect on its reception, as end-users are more likely to use the software product if it is available their local language and dialect.

In order to facilitate the process of localizing applications, a wide array of software internationalization practices has been established. A common practice is the use of key/value pairs, where one key corresponds to one value for each language that the target application is being localized in. The pairs are then exported to translation platforms where translators can access them.

Providing an acceptable translation requires information regarding the application context where each key appears in the user interface. This allows translators to consider user interface constraint limitations (for example, a text fragment may fit inside a button in one language, but not in another), as well as detect value ambiguities in their translation (for example, the word ”banco” in the Portuguese language can be translated to either ”bank”, ”stool” or ”bench” in English), among many other relevant factors. One possible approach to meet this requirement is to associate the key/value pairs with a screenshot of the application context they appear in. Several localization platforms allow this association between screenshots and other files to the key/values themselves.

The association process is, however, performed almost entirely manually at present. The user is required to:

- Execute the application manually.

- When text fragments are detected on screen, manually take the screenshot.
- Check the keys to which the values present in the screenshot taken are associated in the application's key/value resource files, and associate the screenshot with each one.

This process is particularly time-consuming and even unfeasible when the number of keys is high or the application's user interface reaches a certain level of complexity.

1.2 Objectives

The central goal of the internship is to create a tool that automates certain portions of the process of obtaining the KVS (Key/Value/Screenshot) associations. The tool works by modifying the target application's source code directly so that when the user executes the application and traverses the User Interface, the associations between key/value pairs and the screenshots are automatically obtained and exported to a localization platform.

The tool implementation is divided into the following components:

- A code transformation module, which takes an application project's source code as input, and exports a modified version (including the injected code fragments) integrated with the platform-specific library which, when executed, reports the key/value pairs and the associated screenshots.
- A set of platform-specific library modules (one module for each supported platform) which are interfaced by the added code fragments in order to obtain the KVS associations.
- An intermediate web module, which is responsible for receiving the obtained associations from the modified application in a dataset, giving the user the option of analysing them and exporting them to a localization platform.

The final product was integrated with WIT Software's i18n localization platform, although it was left in a state where it can be extended to other localization platforms as well, with minimal modifications

to the export module. The application platforms supported are iOS (Objective-C) and Android (Java) and, likewise, the tool can be extended to other platforms and programming languages.

The internship also consolidated the author's academic tenure, and improved his skills and experience in software engineering, producing a satisfactory product to both him and the company.

1.3 Report structure

Aside from the introduction, the document is comprised of the following sections:

- State of the art: The features and shortcomings of some of the current localization support tools and approaches are explored. Possible code transformation tools and frameworks are also analyzed.
- Project management: Details the structure of the project done at WIT. Describes the software development lifecycle used, as well as the success metrics defined for the project.
- Requirements: Details the functional and non-functional requirements of the tool.
- Architecture: This section presents a structural analysis of the tool, its components, and the way they interact with each other.
- Code analysis and transformation: details the approaches used to implement the transform module and support libraries.
- Testing and evaluation: details the tests performed on the tool, and contains an assessment of the string coverage results.
- Conclusion: Analyzes the results and work done during the internship, the challenges faced and the future work. It also contains a more personal note by the author on how the internship progressed.

The document also contains the following appendices, which were created to make the rest of the document more readable:

- Appendix A: The context of this document's internship.
- Appendix B: Viability tests performed on the CodeWorker and ANTLR tools.

- Appendix C: Some relevant implementation details for the tool.
- Appendix D: The web module's REST API documentation, with detailed descriptions of each HTTP method defined.
- Appendix E: First semester Gantt chart.
- Appendix F: Second semester Gantt chart.
- Appendix G: A detailed description of the weekly tasks and work done on each week during the internship.

In addition, the report was submitted along with the tool's user manual as a private annex.

2 State of the art

In this section, an analysis is performed on the state of the art. It is intended to list some of the already existing alternatives and approaches used in the field of software localization support, and analyze their features and shortcomings.

This section also details the research and experimentation work performed on some of the current tools available for code analysis and transformation, in order to list possible solutions for the project's code transformation module implementation.

2.1 Internationalization practices

This section describes some of the approaches used in software internationalization.

Software internationalization is the set of practices and methodologies used to make software products easier to adapt (localize) to different languages and regions. Software localization often has a broader scope than simply translating the UI text fragments. Other factors often need to be taken into account, including:

- Different languages may use a different language orientation to represent text, such as right-to-left (RTL) top-to-bottom (TTB) or left-to-right (LTR) orientation. This often means that the UI layout itself has to be adapted to different markets. [41]
- Certain UI decisions can be appropriate for a given market's target audience, but inappropriate for another. For example, a given color scheme or image may be offensive in certain cultures, or simply not have the desired effect on the end users [45].
- The measurement units can be different in different countries and regions. For example, while most countries in the EU adopt the metric system, the United States of America employs different units, like the pound for weight and the mile for distances. [27]

While the internship's scope does not include the localization of application elements other than textual content (translation), this section also describes some common internationalization practices directed at handling them.

2.1.1 String encoding

In computing, character encoding refers to the binary format by which character strings are represented internally by a machine.

Traditionally, ASCII encoding was used in most software applications up to the advent of Unicode encodings in 1987 [19]. ASCII encoding simply uses one byte to represent each character on the string. While this approach is often sufficient for a number of use cases, it proves insufficient when it is necessary to support languages other than English, as one byte is often not enough to represent the full character set of a given language.

In order to address this shortcoming, Unicode encoding is often used. Such encoding implementations allows for the representation of a larger set of characters by using the necessary number of bytes to represent each character. The most widely used implementation is UTF-8, which allows up to four bytes to be used for representing a character. The encoding also has the added advantage of being backward-compatible with ASCII, as the characters supported by ASCII are represented in the same format with UTF-8, requiring no conversions between the two formats when localization is not a primary concern.

Other formats can be better suited for some use cases. For example, UTF-16 encoding has been shown to require less space when the text contains primarily characters not supported by ASCII (such as Japanese or Chinese characters). While UTF-8 does support these characters, if the text is large and space is an issue, UTF-16 can be a consideration. However, UTF-8 remains more efficient in representing text which is predominantly made up of ASCII characters, which is a general occurrence when processing text for a large portion of the existing languages. On the other hand, the UTF-32 variant has seen very little use in real-world applications due to not being space-efficient, always requiring 4 bytes to store each character [25]. Figure 1 shows the binary representation of different characters in the three encoding formats.

character	encoding	bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001
あ	UTF-8	11100011 10000001 10000010
あ	UTF-16	00110000 01000010
あ	UTF-32	00000000 00000000 00110000 01000010

Figure 1: Character representations in different Unicode encodings. [36]

2.1.2 UI elements separation from source code

UI elements such as images and text should be separated from the application source code, as well as use a different folder or storage for each target language. Figure 2 illustrates the practice on the .NET platform. A different resource folder is used for the English and Italian languages.

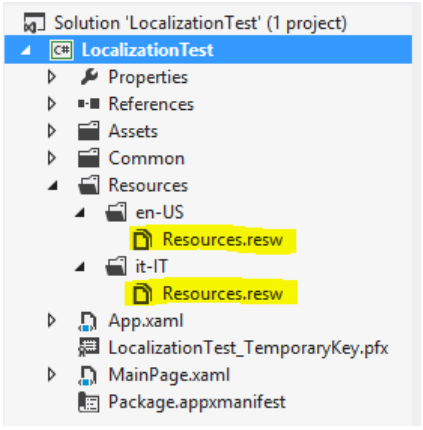


Figure 2: Localized resources on the .NET platform. [35]

Layout definitions such as the on-screen position of buttons, text input fields and other UI elements can also vary according to the language and should, therefore, be stored separately where possible as well. This practice is especially important if the application is to support languages with different text orientations, such as vertical, right-to-left (RTL) and left-to-right (LTR).

Defining the resources separately from the source code where possi-

ble allows additional locales to be supported in future releases with minimal significant source code modifications. [24]

2.1.3 Key/Value approach

When one needs to access the resources stored in the previously described manner, a Key/Value approach is often employed. When retrieving a resource (string, UI element, image or other) from within the source code, the resource's "key" is used to access it. Figure 3 shows an example for a key named "cat_key". The value is "gato" in the Portuguese(pt) locale, "cat" in the English one, and "chat" in the french one.

The same key is mapped to different resources according to the current locale, and therefore this approach gives access to the resources which suit the current locale without any changes to the source code.

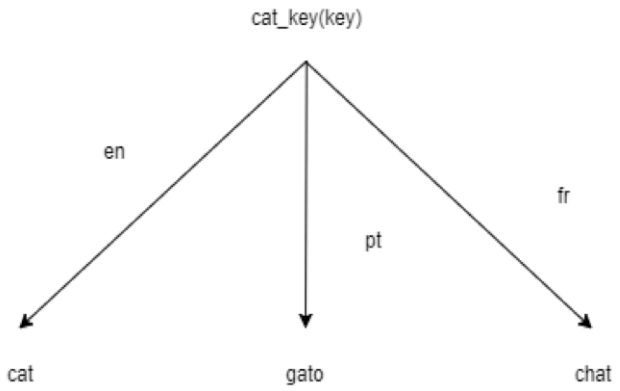


Figure 3: Example of a possible key/value correspondence.

2.1.4 Locale hierarchy

When developing a localized application, it is often not possible to support every regional locale where the application is to be released.

For example, an application may be localized in Brazilian Portuguese (pt_BR), and one may wish to release it in Portugal, where the European Portuguese dialect (pt_PT) is spoken.

Assuming the resources are not available to translate the product in the target locale, a high number of users may nevertheless be reached if the language defaults to Brazilian Portuguese rather than using the

base language (usually English), due to it being more similar to the target locale. On the other hand, if the application is not localized into pt_BR either, the system can simply default to a base language like English until either Portuguese dialect is supported. Other examples of locale hierarchies include the Canadian and French dialects of the French language and the United States and United Kingdom dialects of English.

While users often prefer to use applications localized in their specific regional locale, this practice nevertheless provides a means of releasing applications for a broader audience, with the trade-off of not providing translations with the same level of appeal for users of the target regions.

2.2 Platform internationalization support

In order to make localization less complex for developers, many platforms have features aimed at facilitating the process.

This section details some of these features in the iOS and Android platforms, which the internship focuses in. Similar features are also present in other platforms, however.

2.2.1 Android

The Android Studio IDE allows for the creation of different resources for each supported locale. The resources are stored in a ”res” directory on each module of the project, which contains a specific subfolder for each type of resource (i.e. menus, layouts, textual values, styles and others). Each subfolder contains a different storage for each supported locale. In particular, the values folder contains XML files with simple value definitions, such as strings, integers, and colors used in the application. In the example illustrated in Figure 4, the project would contain different string type resources for each of the five languages listed, while the other resources present in the values folder would remain unchanged for each version.

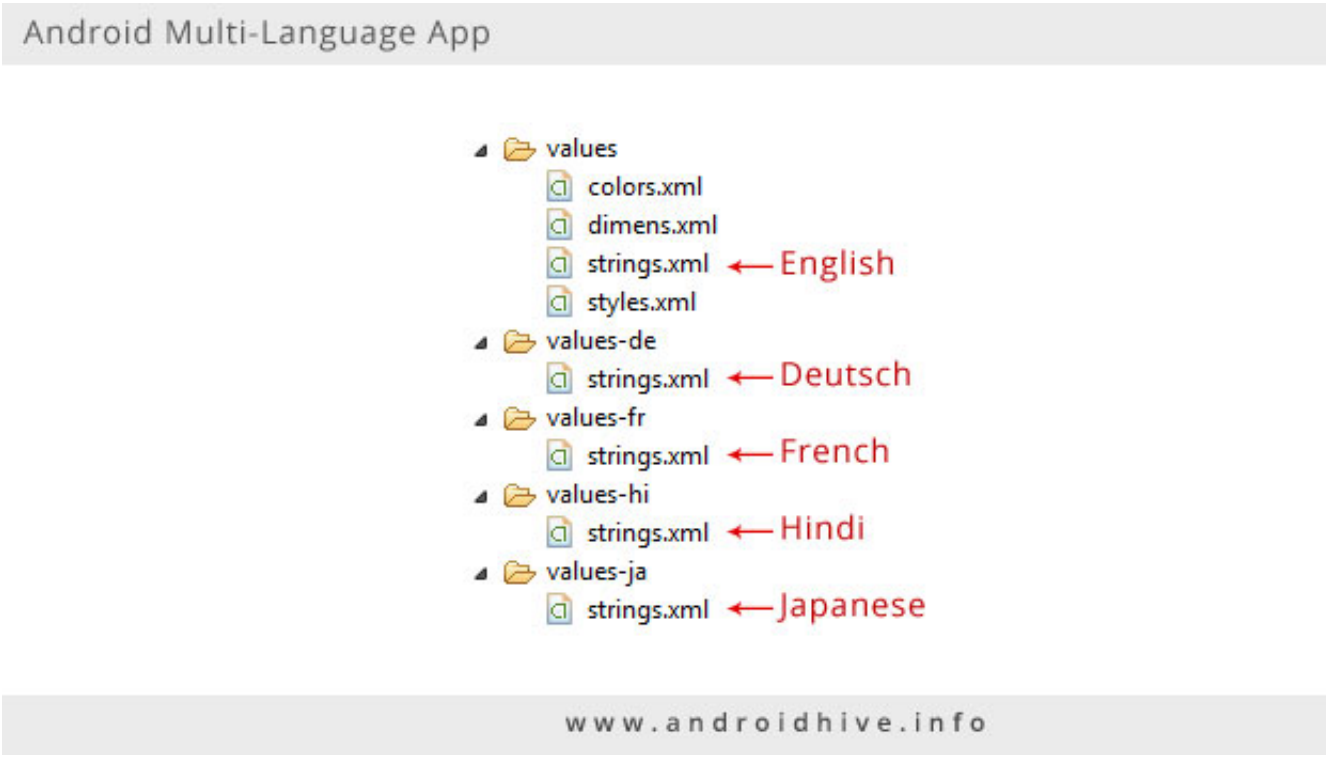


Figure 4: Multi-language Android Studio project values folder. [8]

During project compilation, Android Studio creates a POJO class named ”R”, which saves the resources in static class attributes. The resources can then be accessed from the source code and other project files using the key ID independently from the locale, as shown in Figure 5.

```
@Override
public void showSuccessfullySavedMessage() {
    showMessage(getString(R.string.successfully_saved_task_message));
}
```

Figure 5: Retrieving a string from Java code. [21]

2.2.2 iOS

Similarly to Android Studio, Xcode also has a number of features aimed at facilitating the localization process. The localized resources are placed inside on different folders for each locale. The folders are named <locale name>.lproj and contain the project's string and other resource files for the associated locale. The string files are stored in Localizable string files (Figure 6). The Localizable strings files are internally stored in .lproj folders and contain the localized strings list of the project for each particular locale.

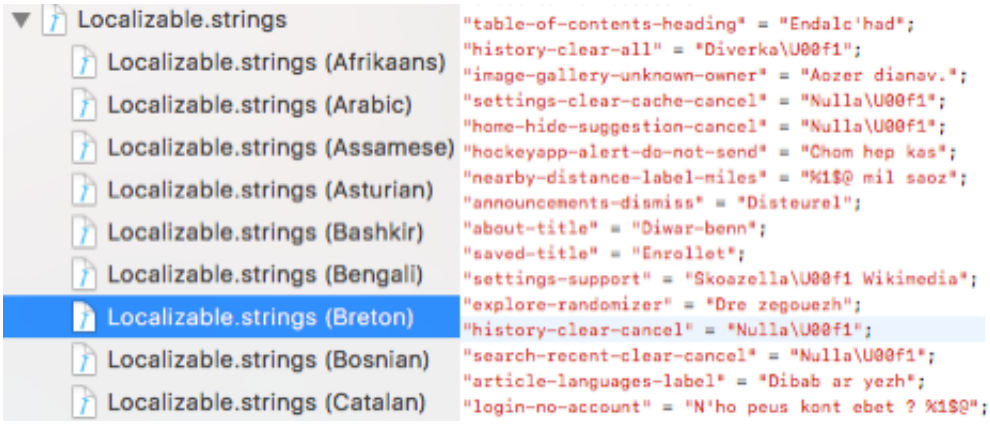


Figure 6: Example Xcode project translated into several idioms. [43]

The developers also have the option of establishing a ”Base” language, which is used as a template during development and uses placeholder strings in the developer's desired language. This allows strings to be stored separately to the layout files (.xib or .storyboard files).

While Xcode supports the definition of layout files individually for each locale, for applications that do not require extensive modification between locales, an auto layout feature is also available which simplifies the process. This feature automatically adjusts the position, size and orientation of the elements in the UI to incorporate strings of different sizes, and even changes their orientation when the text uses a locale with orientation other than the one used in the Base language, such as RTL.

2.3 Localization (l10n) platforms and support tools

This section analyzes some of the present features of various localization support platforms. Such tools aim to support the steps of the translation process. One key aspect that should be mentioned is that the internship aims mainly to provide extra functionality to existing tools, rather than replace their use or compete with them, as no tools were found that contain features which are similar to what the internship aims to implement. Therefore, the analysis focuses on listing some of their notable features and shortcomings, and whether the project could feasibly complement their use.

The desirable features for such a platform would be:

- Utilizes a key/value approach for storing keys.

- Allows the association of screenshots to the key/value pairs.
- Allows for additional file associations to the keys, in order to provide disambiguation info.
- Is feasible to be interfaced with from another application, in order to export the screenshots from the tool.

Similar or related features are highlighted in bold. Other notable features are also detailed for each particular platform. Pootle was selected as an example of a widely used open-source platform. The other tools were selected due to being the only tools found that supported the attachment of files to each key.

2.3.1 Pootle

Figure 7: l10n support tool - Pootle.

Description Pootle is an open-source translation support tool that aims to make the translation process simpler. It allows for the hosting of the translation platform on the user's machine or system, and making it possible for the translating team to access them remotely or locally. [32]

Features Some notable Pootle features include [33]:

- Allows integration with version control systems such as Git and SVN, updating key/value resource files as needed on the projects.
- Allows access to machine translation services which, while insufficient compared to human translations, do provide useful suggestions on the values of keys.

- Translation statistics that make it easier to track the translation process.
- Translation memory, which provides matching translations to a given string if it has appeared before, speeding up the translation process.
- Translation template files, which contain the full list of key/values in the ”original” supported language.

However, Pootle notably lacks the ability to associate files with specific keys out of the box, which makes it unfeasible to integrate with the internship's tool.

2.3.2 Multilizer

Figure 8: l10n support tool - Multilizer.

Multilizer is a proprietary localization support tool. Created in 1995 by Rex Partners Oy, a Finnish technology company, it features a large number of innovative features that make localization easier to perform. The most notable feature is the ability to automate key/value pair loading from a given project, not requiring the user to export them to a platform, unlike other, similar tools. [29]

Features Some notable Multilizer features include [29]:

- Supports machine translation suggestions for translations.
- Allows for automatic scanning of keys from resource files, without input from the user, supporting over 30 file formats. Also allows for format conversions.
- **Allows file associations to specific key/value pairs, including screenshots.**

- Also supports annotations to the pairs, to provide other ways of detailing context.
- Localization templates.
- Translation memory, similar to Pootle.
- Allows for both offline translation management in a single machine, and information exchange on a platform.
- **Can be interfaced from either Delphi (Pascal) or C++, in order to export the correspondences.** [28]

Assuming a license to the product is available, the localization tool could feasibly be integrated with Multilizer, by providing the screenshots and contextual information automatically. This is made easier by the fact that Multilizer supports screenshot associations out of the box and that it can be interfaced from C++ and Delphi.

2.3.3 Transifex

Figure 9: l10n support tool - Transifex.

Transifex is a proprietary localization support tool that aids the localization process not only for software but also other products, such as video subtitles. Based on an online web platform, it features many interesting features, including the possibility of associating screenshots with key/value pairs. [39]

Features Some notable Transifex features include [39]:

- **Support for screenshot association to key/value pairs.**
- Allows interfacing from JavaScript, to upload key/value pairs and other files.
- Translation Memory, similar to Pootle.
- A glossary of each key, detailing its meaning.

- Style guide, to give translators information regarding the company's brand, style, voice and audience
- Team manager, allowing for roles, permissions and subteams to be assigned to members of the translation team.
- Reporting and workflow to track current translation tasks and progress.
- **Can be interfaced through an API, compliant with REST.** [38]

As Transifex can be interfaced with and allows screenshot associations, the tool could potentially be run alongside it, assuming the user possesses a Transifex license.

2.3.4 Text United

Figure 10: l10n support tool - Text United.

Text United is a proprietary localization support tool that allows for both online and local hosting of content.
It contains many of the features of the aforementioned products, including the possibility of associating screenshots to the key/value pairs. [37]

Features Some notable Text United features include [37]:

- Supports both machine and human translation.
- Features Translation Memory.
- Allows for integration with Version Control Networks such as Bitbucket and Github.
- **Allows for screenshot associations with key/value pairs**, as well as comments to a specific translation.
- Features a translation project management system that allows for the creation of subteams, and the assignment of tasks and deadlines.

As it allows for screenshot association, and interfacing with its platform, Text United would also benefit from the tool to automatize the process of obtaining and uploading the screenshots.

2.3.5 WIT Software solution (i18n)

WIT Software currently has its own localization solution, a localization web platform called i18n. This tool supports the usual key/value approach, with the possibility of adding screenshots and other files to each pair, for contextualization purposes.

At present, users are still required to obtain the screenshots manually, which is a very time-consuming process.

2.3.6 Feature comparison and conclusions

	Pootle	Multilizer	Transifex	Text United	i18n
License	Open-source	Proprietary	Proprietary	Proprietary	Internal (WIT)
File/Screenshot associations	No	Yes	Yes	Yes	Yes
Association procedure	N/A	Manual	Manual	Manual	Manual
Integration feasibility	No	Yes	Yes	Yes	Yes

Table 1: Localization tool comparative analysis (relating to the desired features).

After analyzing the features for each tool, the author concluded that:

- Every analyzed tool uses a key/value approach.
- Most of them do allow users to provide contextual information on individual keys/values.
- Screenshot and file association to specific KV's is supported, with the exception of Pootle.
- Every tool analyzed with the association feature requires the user to upload the screenshots himself, and obtaining the screenshots is also done manually.
- No tools were found which performed this task automatically, even locally.

Therefore, at present no tool was found which saves the user the task of obtaining the screenshots for a given application and associating them with the keys. This project focuses on complementing the screenshot export function of these tools, by automatizing the process of obtaining and associating them with the key/value pairs. While the focus

is placed on the WIT platform (i18n), the tool could feasibly be integrated with any of the analyzed tools except for Pootle.

2.4 Code analysis and transformation tools

In this section, several code analysis and transformation tools are explored, with their features and approach being analyzed, in order to be able to determine if they are feasible for the implementation of the project's code transformation module.

In order to make a proper analysis of existing technology, it must first be specified which features are going to be to required.

The features that the tools must contain are as follows:

- Source code analysis must be supported.
- One must be able to detect source code fragments that follow a given pattern.
- Source code transformation must be supported at the given code patterns, or at least be feasible to implement alongside the tool.
- It must feasibly support any programming language (generic tool), or at the very least Android (Java) and iOS (Objective-C).
- It must be feasible to extend the tool with an exporting module/interface or incorporate it with one.
- The tool must be open-source.

In order to implement these features, three different types of tool were considered:

1. Specialized code transformation tools.
2. Parser generators.
3. Static analysis tools.

2.4.1 Parser generators

A parser generator, or compiler-compiler, is a tool that generates a compiler, or parser from a given lexical and syntactic definition. Most commonly, the following definitions are required to create a parser for a given language using such a tool:

- A list of tokens, usually defined using regular expressions (REGEX).
- A grammar (syntactic definition).

The generated parsers commonly create an intermediate representation of the source code, such as an AST or parse tree, which could be used to perform code analysis and transformation tasks.

Lex/Yacc(Bison)

Figure 11: Code analysis and transformation tool - Bison.

Description and features Lex is a parser that processes tokens from program input, using regular expressions. YACC is a tool generally used to build compilers for existing or new programming languages that allows C code to be executed when any syntactic rules are matched. Lex and Yacc are both part of the Bison parser generator, which takes the .lex and .yacc file and generates C code for the given parser.

Advantages

- Gives complete freedom regarding the definition of the data tools to store the syntactic data. One could choose to use an AST, parse tree or any other appropriate structure.
- Assuming the grammars are properly defined, it could feasibly be used on any language and platform.
- Allows the transformation of the tree structure freely, either changing or adding nodes before regenerating code.

- Fully open-source.
- The intern has experience with the tools from the Compilers course in his Bachelor's degree.

Limitations

- Provides limited functionality out-of-the-box.
- Requires manual definition of the grammars for the language, or a subset of it.
- Robust grammars are not as readily available online as ANTLR.
- Mostly superseded by similar newer alternatives, like ANTLR.

ANTLR

Figure 12: Code analysis and transformation tool - ANTLR.

Description and features ANTLR is a powerful parser generator that can be used to read, process, execute, or translate structured text or binary files. It is widely used in academia and industry for a multitude of applications, including the implementation of new languages, tools and frameworks. Since version 4.0, it internally builds a parse tree rather than an AST. [50]

Advantages

- Very popular tool, with a large amount of documentation and code examples available online, as well as grammars.
- It is written in Java and available from repositories such as Maven (Java) and Pip (Python), allowing for easy integration in a project.
- Can be interfaced from other languages, such as C++ and C#.
- Allows for easier processing of specific grammar rules, using visitor and listener class definitions, in an event-oriented paradigm.
- Also supports inline code transformations inside the grammars themselves.

- Contains plugins for several IDE's, including Eclipse, Netbeans and IntelliJ.
- Fully open-source.

Limitations

- The generated parse tree was not designed to support transformations. The process is possible but not as intuitive as with other tools. In order to perform the transformation, a listener/visitor could be used to traverse the tree and make the necessary changes with a TokenStreamRewriter. ANTLR also supports code transformations by adding text directly within the grammars after the tokens.

2.4.2 Static analysis tools

Static source analysis can be defined as the analysis of source code without actually executing it, and detecting certain, predefined code patterns according to a given set of rules. Many approaches for such a tool exist, including: [44]

- Unit level: Analysis that takes place within a specific program or subroutine.
- Technology level: Takes into account interactions between unit programs to get a more broad view of the overall program, which helps avoid false positives in the pattern detection.
- System level: Takes into account the interactions between unit programs, but without being limited to one specific technology or programming language.

Static code analysis is commonly used by IDE's, in order to provide useful feedback to programmers, as well as various vulnerability and security defect detection tools.

The only static analysis tool found that supports both Objective-C and Java, was Facebook Infer.

Facebook Infer

Figure 13: Code analysis and transformation tool - Facebook Infer.

Infer advantages

- Supports both Objective-C and Java.
- Fully open-source.
- Contains a multitude of features for static code analysis, including the possibility of building custom checkers, which could be used to find the needed code patterns.

Infer limitations

- It was primarily designed as a tool to find bugs and security vulnerabilities in source code, which is not the desired use case.
- Documentation and code examples are limited, particularly in the checker's portion.
- The checkers are written in the OCAML language, which the intern has no experience in.
- Testing new checkers requires recompiling the Infer distribution each time, making it a very-time consuming process in itself.
- The checkers use a CFG approach, and not a tree, providing less flexibility in their manipulation.

2.4.3 Specialized code transformation tools

These tools are created for the specific purpose of modifying the source code. Many different approaches can be used, like analyzing and changing the AST of the code, or use a grammar combined with regular expressions to detect the necessary code patterns, and a different

one to define the new fragments to be injected.

In this section the analyzed tools that use this approach are detailed.

CodeWorker

CodeWorker is a versatile open-source parsing and code transformation tool designed around the principles of generative programming. It allows for several approaches to code analysis and transformation, including the definition of code templates, in-place code transformation and more. It uses its own DSL (Domain Specific Language), for which it includes a built-in interpreter, and can also be interfaced from C++, Java and .NET [49].

CodeWorker advantages

- Very versatile tool, particularly in regards to the code transformation portion.
- BNF based approach makes it relatively simple to learn [48].
- Has a built-in interpreter for executing its language-specific commands, and also allows interfacing from C++, Java and .NET.
- Fully open-source.

CodeWorker limitations

- While documentation is fairly thorough, its usage is limited, making it difficult to find code examples on the Internet.
- The project has been discontinued, and minor incompatibilities have been reported on recent versions of Windows. [47]

DMS Software Reengineering Toolkit

Figure 14: DMS Software Re-engineering Toolkit - company logo (Semantic Designs).

DMS SRT is a broad set of tools for source code analysis and transformation, developed by Semantic Designs. Allows for relatively simply source code re-engineering for a multitude of programming languages.

DMS SRT advantages

- Very versatile tool, allowing for both analysis and transformation, supporting lexical, syntactic and semantic approaches.
- Supports Objective-C and Java, as well as a multitude of other technologies.
- Does not require the definition of grammars or rules for the languages it supports.

DMS SRT limitations

- Proprietary software.
- Limited online documentation and source code examples.

Spoon

Spoon is an open-source library that enables the analysis and transformation of Java source code. It provides a complete and fine-grained Java metamodel where any program element (such as classes, methods, fields, statements and expressions) can be accessed both for reading and modification, using an event-oriented approach. It operates using the source code's Abstract Syntax Tree (AST).

Spoon advantages

- Very versatile tool, allowing one to easily modify the AST Nodes, or adding new ones, and then generating the new code again.
- Code fragment context can be easily accessed, due to the programming model used.
- Fully open-source.

Spoon limitations

- No support for Objective-C, or languages other than Java.

2.4.4 Code transformation tool analysis

This section analyzes the code analysis and transformation explored in the state of the art, and attempt to choose the solution to implement the transformation module.

Use a specialized code transformation tool This option consists of using an already implemented tool to scan the source code, detect certain patterns and make the necessary changes.

The tool would need to support, at the very least, both Java and Objective-C, although it would be desirable to have it be extensible to other platforms as well.

Build a new parser for each platform Such an approach would involve writing the grammar specification for a subset of the target platform's language, in the chosen parser generator's grammar syntax, or otherwise obtaining the definition for it.
While this approach would likely incur extra effort and time, it would also provide a higher degree of flexibility. One could, for example, obtain parse trees (or AST's) for any given language, and edit its nodes before regenerating the code, while keeping track of any needed part of the context where the key was mentioned (such as checking if it was called from a method that shows it on screen). Language agnosticism would also be a non-issue.

Considered tools for this approach:

- ANTLR
- Yacc/Bison

Modify an existing static analysis tool Using this approach, the tool selected needs to be able to analyze source code for both iOS (Objective-C) and Android (Java). Many static analysis tools are also designed to simply detect bugs and vulnerabilities on the source code before executing, and using them for this use case would require modifying them, or adding different rules to the analyzed code fragments, such as detecting calls to a method with the given keys.
The only such tool found that supports both languages is Facebook

Infer.

Table 3 shows a comparison between the relevant features of each tool.

	ANTLR	Bison	CodeWorker	DMS Toolkit	Infer	Spoon
Analysis	Yes	Hard	Yes	Yes	Yes	Yes
Transformation	Yes	Hard	Yes	Yes	No	Yes
Language agnostic	Yes	Yes	Yes	No(supports many)	No	Only Java
Extensible	Yes	Yes	Yes	No	Hard	Yes
Open-source	Yes	Yes	Yes	No	Yes	Yes

Table 2: Code analysis and transformation tool comparison.

While Spoon itself is not a viable solution for the project (due to only supporting Java) it provides an interesting approach to analyzing and transforming code and can be a good starting point on the architecture of the Code Transformation model if the parser definition option is chosen. The main issue would be to achieve language agnosticism.

ANTLR would be the most suitable tool for the approach, by making code transformations (via adding tokens) within the grammars themselves. While this approach would require the definition of the grammar rules for each platform manually, ANTLR does have the grammars already defined online for both Objective C and Java in the ANTLR GitHub repository [10], and one would only need to add support for code analysis and transformation to them.

If modifications need to be performed to languages that are not defined in those grammars out of the box, ANTLR would also allows for the definition of the grammar for the subset of the language that is needed in order to apply the transformations.

This approach does not fully complete the necessary analysis and transformation, as it is required to not only find where the keys are being used but also determine if they are being shown on-screen in the application at that point in the source code. This requires a further syntactic analysis. In ANTLR, this is possible via using a ParseTreeWalker and Listener class.

In light of this, the tools chosen for further testing were CodeWorker and ANTLR, with ANTLR (Java) being chosen for the project in the end.

The tests performed are detailed in Appendix B of this report, complete with code samples.

3 Project management

This chapter describes the development lifecycle used for this project, the tools and methodologies used to track it and finally the success metrics defined for its successful completion.

3.1 Software development lifecycle

The choice of a development lifecycle for any particular software project is a widely debated topic in the industry. It should take into account several factors, particularly the nature of the software requirements. If the requirements are set in stone from the beginning, Waterfall or a similar method is often the correct choice. Likewise, if the requirements are uncertain, or bound to change, a more agile method should be employed.

In the case of this particular project, the requirements were quite uncertain and required extensive exploration of the platforms and their user interface structures to determine the time needed to complete implementation and other development tasks. Furthermore, at the start of the project, the intern had limited experience with the technologies being used, further making specific deadlines difficult to establish. For this reason, an agile lifecycle was used. The actual method is not defined by any of the most common agile methods but instead uses several features of different methods. Some of these features include:

- The planning of the traditional phases of software development such as requirements analysis, architecture definition, development, documentation and testing were planned at a high level at the start of the project.
- Weekly meetings were held between the intern and the tutor/supervisor at WIT-Software to provide feedback on project status and what tasks to prioritize next.
- Since estimates were difficult to perform, the project was flexible in nature, with requirements and tasks changing according to the progress made, as well as developments and issues encountered.

3.2 First semester tasks

In the first semester, the following tasks were established at the beginning of the internship:

- State of the art analysis.
- Analysis of tools to support translation and code analysis and transformation tools.
- Architecture definition.
- Exploration of sample Android applications, adding code fragments manually in order to handle the process of obtaining and exporting keys to the intermediate platform.
- Implementation of some Android code transformation features. This feature was partially anticipated due to higher time availability to dedicate to the internship by the author (it was originally planned for the second semester).
- Intermediate internship documentation.

3.2.1 Completion assessment

The first semester Gantt chart is shown in Appendix E, and describes the timeline of the work done.

All of the first semester tasks were completed successfully. While small issues did arise during the implementation of the Android support library (particularly with the handling of toolbar menus), they were eventually resolved.

3.3 Second semester tasks

After an assessment on the work done during the first semester, it was concluded that coverage had been sufficient in the Android applications tested to move forward with the project. Therefore, the tasks established for the second semester were as follows:

- Implementation of a simple User Interface for the tool.
- Implementation of the i18n platform export module.
- Implementation of the iOS support library.

- Exploration of sample iOS applications implemented in Objective-C, adding code fragments manually in order to handle the process of obtaining and exporting keys to the intermediate platform.
- Implementation of the iOS code transformation module for Objective-C iOS projects.
- Tool validation and improvement after assessment on WIT projects.
- Final internship documentation.

3.3.1 Completion assessment

The second semester Gantt chart is shown in Appendix E, and describes the timeline of the work done.

During this semester, most of the tasks were completed successfully. Some notable challenges faced during this semester include:

- The author had no prior experience with iOS, Objective-C or Swift, and very limited experience with Apple technologies in general. Efforts were made at the beginning of the semester to get better acquainted with the technologies.
- The disambiguation process was accessed as resulting in usability issues traversing the UI. Therefore, a new approach had to be devised (Appendix C.2).
- Performing code analysis and transformation procedures on Xcode pbxproj files was very complex, mostly due to the unavailability of official documentation on the structure of these files. Transformation patterns were established from studying files for existing projects and from unofficial online sources.
- The web interface was accessed as having room for improvement from an aesthetic point of view. Efforts were done to improve it.

In addition, it was expected that an assessment of the tool's features was performed on a WIT-Software project, but, due to time constraints, this demonstration was only performed on open-source projects.

3.4 Success metrics

In the interest of evaluating the success, or lack thereof, of a software project, it is common practice to establish a list of metrics used to

evaluate the result.

In the case of this project, the ideal scenario can be defined as being able to report every key correctly, for any given situation where a key is presented in the user interface. This goal is not likely to be achieved, as user interface components for each platform are very diverse, and often rely on third-party libraries that cannot be realistically handled completely by the tool (for example, due to the use of native platform structures that cannot be detected by simply analyzing the source code easily).

Therefore, a more realistic goal is to cover as many keys as possible, for a given set of applications. Even if the developer is unable to obtain every key present in the user interface with the help of this internship's tool, it is still better for him to only be required to take the manual route for a small number of keys in isolated scenarios. This key coverage percentage is covered in the coverage evaluation section of the report. Keys which do not appear in the user interface, and are only used within the code for other purposes (or not used at all) are ignored.

However, simply covering as many keys as possible is not enough, as it is also not desirable to report keys that are not currently shown on screen. This occurrence is known as "false positives", and minimizing them as much as possible is another goal. Consequently, the number of false positives reported on each tested application will also be tracked. These evaluations were performed separately for each target platform, as it is possible to have differing results in each one.

4 Requirements

This section lists the implementation requirements of the project. The defined requirements are divided into the following categories:

- Web functional requirements (ID prefix: FR_WEB): refers to requirements related to the features of the web module.
- Transformation module functional requirements (ID prefix: FR_TR): refers to requirements related to the code analysis and transformation process.
- Support library functional requirements (ID prefix: FR_SL): refers to requirements related to the features of the platform-specific support libraries.
- Non-functional requirements (ID prefix: NFR).

The list underwent several revisions during the internship, and the requirements shown represent the final version.

4.1 Functional

ID	Description	Dependencies
FR_TR_1	Receive a zipped project from the web interface	N/A
FR_TR_2	Read the application name and version from the project configuration files	FR_TR_1
FR_TR_3	Read the localizable strings from the project's string files	FR_TR_1
FR_TR_4	Remove any ambiguities from the key/value pairs, and save them in both the project, and the database	FR_TR_1, FR_TR_3

FR_TR_5	Execute the necessary code transformation steps to interact with the support library	FR_TR_1, FR_TR_3
FR_TR_6	Make any other necessary changes to inject the support library in the project and include it in the build	FR_TR_1
FR_TR_7	Return the modified application to the user in a zipfile	FR_TR_1, FR_TR_3, FR_TR_4
FR_WEB_1	Display the homepage	
FR_WEB_2	Send a project to be transformed	FR_WEB_1, FR_TR_1, FR_TR_7
FR_WEB_3	List datasets according to the application, platform and version	FR_WEB_1
FR_WEB_4	Display a dataset's contents, including the covered keys and the screenshots obtained for it	FR_WEB_1, FR_WEB2
FR_WEB_5	Delete a dataset	FR_WEB_1, FR_WEB_2
FR_WEB_6	Login to i18n	FR_WEB_1, FR_WEB_2
FR_WEB_7	Export a dataset to an i18n project, deleting every screenshot already present	FR_WEB_1, FR_WEB_2, FR_WEB_6
FR_WEB_8	Export a dataset to an i18n project, only adding screenshots to strings that do not have one appended	FR_WEB_1, FR_WEB_2, FR_WEB_6

FR_WEB_9	Export a dataset to an i18n project, adding screenshots to every key convered	FR_WEB_1, FR_WEB_2, FR_WEB_6
FR_SL_1	Obtain the application's name and version at runtime	N/A
FR_SL_2	Obtain the list of localized key/value pairs at runtime	N/A
FR_SL_3	Detect localized string occurrences in the device's screen, according the use cases defined	FR_SL_2
FR_SL_4	Detect whether a given key has already been reported in the given context	FR_SL_1, FR_SL_2, FR_SL_3,
FR_SL_5	Capture screenshots during the occurrences, saving them in the device's storage	FR_SL_3
FR_SL_6	Save the key/value/screenshot associations in a JSON file in the device's storage	FR_SL_3, FR_SL_4
FR_SL_7	Detect the occurrence of the platform-specific export trigger event	N/A
FR_SL_9	Export a dataset to the Web Module via HTTP in a zip file, optionally providing the dataset name	FR_SL_1, FR_SL_2, FR_SL_3

Table 3: Functional requirements.

4.2 Non-functional

ID	Quality attribute	Description
NFR_1	Scalability	The web server should be able to handle multiple requests at the same time.
NFR_2	Scalability	Transformation requests should be supported for zipped files up to at least 300MB.
NFR_3	Compatibility	The iOS support library should not require a version over 8.0 as the minimum version.
NFR_4	Compatibility	The Android support library should not require a version over 21 as the minimum SDK version.
NFR_5	Compatibility	Android Gradle code analysis and transformation should support at least Gradle versions 2 and 3.

Table 4: Non-functional requirements.

5 Architecture

This section describes the architecture of the overall system where the tool is integrated into, as well as a more detailed description of each component and the way they interact with each other. Such a definition is useful not only to plan ahead on what will have to be implemented and how, but also to help any future developers understand how the system works, and what steps should be taken in order to potentially extend its features in the future. Due to the agile nature of the project, the architecture was subject to frequent changes and updates.

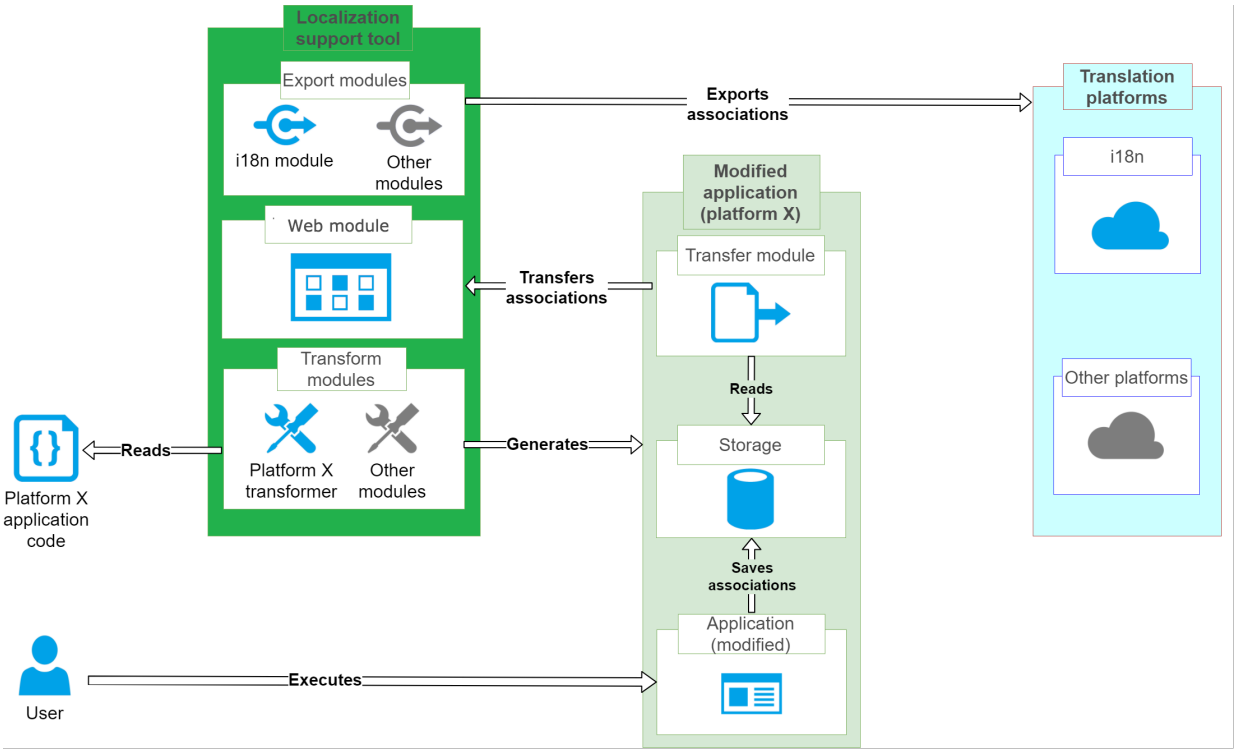


Figure 15: Overall system architecture.

Figure 15 illustrates the interactions between the components of the system. The basic flow of actions to obtain the associations for a particular application is the following:

- The user uploads a project through the Web interface.
- The code transformation module creates a modified version of the project.
- The user builds and executes the modified application in its target platform.

- As the user manually traverses through the application's UI, whenever key/value pairs are detected on screen, the modified application makes calls to the platform-specific library to save them in the device's storage.
- After the user triggers an event (defined for each platform individually), the associations are exported to the Transfer Module (localization tool) as a dataset.
- The user can then view the exported datasets in the export module's Web Interface, and export them to a supported localization platform.

The following subsections describe the structure of the components in detail. It should be noted that the translation platform was already implemented before the internship, and was not a part of the work done by the author, but rather integrated with the rest of the system.

The programming language used across most of the project was Java, except where noted otherwise. Java was chosen due to the author's familiarity with the language, as well as the assessment that the code transformation tool (ANTLR) had higher quality documentation available for its Java API.

5.1 Transformation module

This component is responsible for modifying the application project's source code and any other necessary files in order to obtain the associations between screenshots and key/value pairs at runtime. The module is divided into a platform-specific module, and a generic module.

The transformations consist of the analysis of the source code of the application and the addition or replacement of code fragments following specific patterns, as well as the injection of the platform-specific support library. The added code fragments interact with the library in order to execute the necessary reporting steps defined in the support library section. In order for the library itself to be included in the modified project, it is also necessary to transform the project configuration and/or build files to include it.

In order to apply the transformations correctly, a set of pattern rules was defined for each platform. The rules define which code portions need to be changed and how.

The module is also responsible for retrieving the application's name, version and list of key/value pairs and saving them in the persistence module's database.
The technologies used by this module include:

- ANTLR4, for performing static code analysis and transformation on program source code, as well as configuration and string files when necessary.
- JDOM2, for analyzing and transforming project XML files, such as plist files in iOS, and string resource files in Android.

5.1.1 Generic transformations

Some basic transformation operations are common for every platform and were implemented in an abstract Java class from which the "transformer" classes for each platform extend, in order to alleviate the task of extending the tool to other platforms. These include:

- Obtaining a list of project files with a given file extension (example: .java, .m), to which the platform-specific transformations are applied.
- String ambiguity handling, after reading the localizable strings from the project (platform dependant).
- Saving the application's name, platform and version in the database, along with the original and modified strings.

5.1.2 Platform-specific transformations

These transformations are platform-specific, and were implemented in subclasses of the base transformer class. They are responsible for the following tasks:

- Obtaining a list of the key/value pairs, from the project's resource files. Platforms often use different formats to store them, so this will require a different implementation for each particular one.
- Analyzing and modifying the source code, inserting the necessary code fragments to interact with the platform-specific library

and obtain the associations between the key/value pairs and the screenshots. In order to detect the needed patterns in the source code, ANTLR listeners were used, along with the "ParseTreeWalker" class. Changes were performed inside the listener definitions, using the "TokenStreamRewriter" class.

- Other tasks, such as key/value modifications in the resource files to eliminate ambiguities and changing configuration files to include the platform's support library.

Implementation details for each particular platform and use case are detailed in the code analysis and transformation section.

5.2 Modified application project

This is the output of the code transformation process. This application project contains the original application's source code and other project components, with the following changes:

- The platform-specific library was added to the project's structure.
- The build and other relevant configuration files were transformed in order to include the library in the build.
- Source code modifications were added, which interact with the library in order to detect key/value occurrences in the UI, take the screenshots, and export them to the intermediate interface module.

5.3 Platform-specific libraries

The platform-specific libraries are meant to aid the process of detecting key/value pairs in an application, capturing the screenshots, and reporting the associations to the web server as a dataset. They are accessed from the modified application through the added code fragments and injected into the project build by the transform module.

The libraries are responsible for:

- Obtaining the list of key/value pairs from the project's resource files.
- Detecting key/value occurrences in the application's UI.

- Obtaining the screenshots of each occurrence and saving associations to the key/value pairs.
- When a predefined user input event occurs, export the associations to the tool's HTTP interface.
- Preventing keys from being reported more than once in the same context.

The libraries are platform-dependent. During the internship, the Android and iOS libraries were implemented, with support given to the Java and Objective-C languages.

Implementation and technology details for each library are presented in the code analysis and transformation section.

5.4 Web module

This module is used as an intermediate interface between the modified application, running in the target platform, and the localization web platform. It is made up of the following components:

- A web interface that acts as the tool's User Interface.
- An HTTP interface compliant with REST that handles requests from both the transformed applications and the web interface.
- A persistence submodule, which saves data regarding each transformed application and the association datasets obtained for it.

5.4.1 Web interface

The web interface contains the following sections:

- Home: the entry point of the interface. Contains general information regarding tool usage and features, as well as links to the other sections.
- Datasets: the menu where uploaded datasets can be analyzed in detail and exported to a supported localization platform.
- Transform: the menu where the user can upload a project and obtain the modified version.
- Help: displays a tutorial for using each of the tool's features.
- About: displays information regarding how the author can be reached out for support and the tool's copyright notice.

The interface was implemented using plain JavaScript and HTML, along with jQuery and Ajax calls to the HTTP interface where necessary.

A more thorough description of the interface's usage and menus, is detailed in chapter 6 of the user manual, complete with screenshots.

5.4.2 HTTP interface

The HTTP interface handles requests from both the modified applications and the tool's web interface, acting as the communication link between the other modules. The following features are implemented:

- Uploading datasets from the modified applications, saving them in the file system and storing information about them in the SQLite database.
- Listing the datasets obtained for a particular application and version.
- Viewing dataset key coverage and other details for a particular dataset.
- Exporting dataset associations to a supported localization platform using the platform's export module.
- Applying transformations to a project from a supported platform, returning the modified project.

The HTTP interface makes use of the Spark micro web framework for Java to host the Restful services. Spark is a lightweight framework designed to facilitate the process of hosting web services as much as possible, requiring no set up outside of installing the framework itself and allowing the services to be implemented with minimal boilerplate code [22]. Unlike most web frameworks available, it supports execution without the use of a web container, using an embedded Jetty server.

The author opted for Spark to implement the HTTP interface because the features implemented in the web portion of the tool do not justify the use of alternatives like Java EE, which require additional configuration steps and incur a large amount of extra implementation overhead for small web interfaces.

The tool was implemented with support for hosting in both web containers like Tomcat and Wildfly (deployable WAR), or as a standalone java application (simple JAR file), using Spark's embedded Jetty web server. Python scripts were created to facilitate firing up and configuring the server in either mode as detailed in the user manual.

A detailed description of the REST API implemented by this module is present in Appendix D.

5.4.3 Persistence submodule

This submodule is responsible for storing and accessing the data regarding the applications, their versions and the datasets obtained for each version.

A dataset contains a list of screenshots, as well as the key/value pairs detected on each one, for a given upload.

The datasets themselves are stored in the file system, within a "datasets" folder in the directory where the tool is stored. Each dataset folder contains a JSON file with the correspondences between the datasets and the screenshots it contains, and the actual screenshots.

In order to store other information, such as the list of key/value pairs detected in the project during the transformation process and the location of the datasets associated with each one, an SQLite database file was used, stored in the tool's root folder. The entity-relationship diagram for the database is shown in Figure 16.

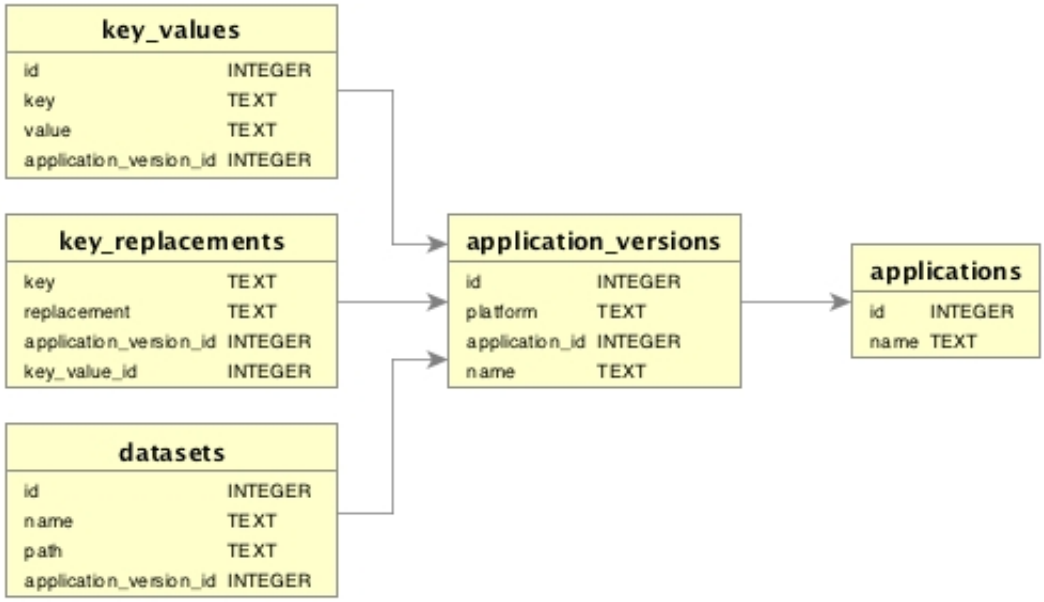


Figure 16: SQLite ER diagram.

Each dataset is associated with a specific version and platform of a given application. This allows the user to obtain datasets for different versions of the same application, or various platforms and device versions.

SQLite was chosen over alternative database engines, because it allows the tool to be fully self-contained, not relying on client-server based database engines like MySQL or PostgreSQL.

5.5 Export module(i18n)

The i18n export module accesses the i18n platform via HTTP calls and requires the user to authenticate with his or her i18n credentials. The following export modes are supported:

- Delete all screenshots from i18n for the given project before appending the ones on the selected association dataset.
- Only append screenshots to keys that do not have at least one yet.
- Append the screenshots to every key covered in the dataset.

The screenshots are appended to the corresponding key/value(the key value is created in i18n if it does not exist). A text file containing the original keys before the disambiguation process is also added for user clarity.

5.6 Translation platform(i18n)

This web platform contains a storage of key/value pairs, and can be accessed remotely via a REST interface. The user is able to:

- Add a key/value pair on a project, for a given language.
- Associate files to existing key/value pairs (screenshots or otherwise).
- Delete or modify existing key/value pairs on a project, for a given language.

This portion of the system is not part of the author's work during the internship.

6 Code analysis and transformation

This section describes the approach used to implement the features of the transform module and the platform support libraries, along with the challenges faced and how they were overcome.

6.1 Transformation patterns

The approach used in both of the supported platforms is quite different, due to the nature of the programming languages and the environment. Three main code transformation patterns were used across the project.

Reflection approach (method swizzling) The implementation of the platform's methods is changed at runtime to achieve the desired behavior by the support library itself, and the transformation module will simply inject the library into the project and initialize it to perform the reflection changes. This approach is the cleanest option, but is not feasible for platforms with static programming languages, like Java, as they do not possess the reflective features required to change code behavior at runtime in this manner.

In order to understand the approach, one must first understand the difference between statically and dynamically typed programming languages at handling method and function calls and definitions.

In the case of static languages like Java, C and C++, all functions and variables are defined at compile time and cannot be changed when the program is executed under normal circumstances.

On the other hand, dynamic languages like Python define symbols dynamically, allowing for their redefinition at runtime. For example, the Python code shown in Figure 17 is fully valid, and replaces a method implementation with another. Any subsequent calls to the method will use the replacement version, rather than the original one.

```
class MyClass(object):

    def my_method(self):
        return 'Hello'

#instantiate the class
my_obj = MyClass()

#save the reference to the original method
orig_method = MyClass.my_method

#swap the instance method "my_method" with a lambda call
MyClass.my_method = lambda obj: orig_method(obj) + ' and goodbye!'

#prints "Hello and goodbye!"
print(my_obj.my_method())
```

Figure 17: Will print ”Hello and goodbye!”

Such a programming pattern is called ”method swizzling” and is particularly useful for operations like logging or code instrumentation.

This approach was used to handle the iOS use cases.

Event driven approach Minimal code fragments are added to the application in specific portions. The fragments call a support library function which creates event listeners that detect whenever the screen's textual content changes. Whenever the listeners detect textual changes in the UI, any keys present are reported. This approach is similar to the reflection one, but requires slightly more transformations to the source code and does not allow the coverage of all use cases, as the platforms do not support event listeners for all the instances where text will appear on screen.

This approach was used to handle most Android use cases that occur in the main View hierarchy, such as TextViews.

Class wrappers ”Container” classes are implemented in the support library for UI structures which are usually instantiated in the source code directly. These classes will have equivalent methods to the original class, with the exception that when methods that show text on screen are called, the screenshot is taken and the key is associated with it. The transformation module is responsible for changing the occurrences of the original classes with their ”wrapped” version. This approach was only used as a last resource, due to being more error-

prone and not handling cases where the UI structures are imported from an external library, rather than being present in the source code.

This approach was used to handle Android use cases that occur outside the main View hierarchy, such as "AlertDialogs" and Menus.

6.2 Android (Java)

The Java programming language is statically typed, and only has minimal reflection features, such as class method listing and access modifier changes (i.e. changing a private attribute or method to public). This makes the reflection approach unfeasible, and therefore the event driven approach was used where possible, with class wrappers handling the remaining use cases covered. Event listeners were created by inserting code fragments on Android Activity method implementations in the project's source code.

More information about the supported Android use cases in the Android platform can be found in Appendix C.1.

6.2.1 Other transformations

In order to inject the library into the project, the .aar file was copied into the project's root folder, and the Gradle file was transformed to include it in the build.

Since the ANTLR grammar repository did not contain a grammar for the Groovy language as of the writing of this report, a modified grammar was created based on the Java grammar with the bare minimum rules needed to perform the changes.

6.3 iOS (Objective-C)

In iOS, method swizzling was used for all the use cases.

Objective-C is a strict superset of the C language, with any valid C source code also being valid in Objective-C. However, it contains support for classes and other object-oriented features, which are implemented on top of the C language using function pointers. Classes in Objective-C can, therefore, in a general sense, be thought of as C structs where each method is actually a function pointer member of the struct it belongs to. This means that, while C-like functions in

Objective-C are statically typed (as is the case in the C language), class method references can be swizzled in the same pattern shown in Figure 17. This extends as far as the native iOS methods defined in the standard iOS libraries, which can also be swapped with others freely. [30]

Method swizzling effectively makes it possible to handle any method defined in Objective-C (either by the standard libraries or on the application code) as an event. For example, if one needs to detect whenever a UITextView changes its textual contents and create a screenshot of the new text, the following process can be performed:

- A new method, called ”setText_LOCALIZATIONHELPER” is created for class UITextView using a category, as shown in Figure 18.
- When the class loads, the method shown is swapped by the original ”setText” method in the class using method swizzling.
- Whenever the ”setText” method is called on the source code, ”setText_LOCALIZATIONHELPER” will be called instead, and the view’s textual contents will be reported.

```
-(void)setText_LOCALIZATIONHELPER:(NSString *)text{
    //calls the original setText method, which is now associated with the
    //setText_LOCALIZATIONHELPER selector, rather than the original one
    [self setText_LOCALIZATIONHELPER: text];
    //report the contents of the view
    [LocalizationHelperiOS.sharedInstance reportViewKeysWithView:self];
}
```

Figure 18: Method to be swapped by ”setText”.

In order to apply the swizzling operations to the project, the only requirement is to inject the library and import it, effectively making the actual source code fragments inserted by the transform module a single library import statement to any file containing the ”main” function in the project. This approach has several advantages compared to the ones used for Android, including:

- Less error-prone.
- Unlike code insertions, it does not run the risk of breaking code integrity when errors do occur. Incorrectly inserting code fragments can prevent the application from compiling.

- Since Objective-C and Swift code use the same runtime, any changes made to Objective-C classes will also be applied to Swift. While the main objective of the internship is to support Objective-C, supporting applications partially written in Swift is nonetheless a welcome bonus.

The swizzling process was applied by implementing the "load" method on each class using an Objective-C category. This method is called on the very first time the class is used by the program. This ensures that the swizzling operations are applied to them and all their subclasses before the first time they are used. [11]

Appendix C.3 details the method swizzling operations performed on native iOS methods that allowed the screenshots to be obtained.

6.3.1 Other transformations

This section describes other transformations performed on iOS projects.

Support library injection In order to inject the support library into Xcode projects, the CocoaPods dependency manager was used. CocoaPods is a dependency manager which allows for the easier inclusion of external libraries in an Objective-C or Swift Xcode project. It uses a file named "Podfile", which is created in the project's root folder, and contains the required libraries (pods) to include in each of the project's targets in a syntax based on the Ruby language. The libraries may be imported from external repositories in GitHub, or locally, by providing a relative path to their location in the file system.

Options other than CocoaPods were considered, but ultimately rejected:

- Injecting the library directly into the project as a static library was abandoned due to being a less clean solution, as the pbxproj files are very complex to analyze and transform, and were generally only modified when strictly necessary.
- Carthage, another dependency manager for Xcode projects, was rejected due to not allowing the inclusion of local libraries, requiring them to be imported directly from GitHub. This was not an option, as it required exposing the library's source code to the project, rather than the compiled version.

The transform module injects the library into the project by copying it to the project's root folder and modifying or creating the project's Podfile to include it. In order to make the necessary analysis to the files, the ANTLR Ruby grammar present in the ANTLR grammar repository was modified [10].

Settings.bundle injection In order to implement the user event which will trigger the export process for the KVS associations detected, the application's Xcode settings menu was modified. This required the modification or injection of the Settings.bundle package in the project, depending on whether the project already had a settings bundle. When it is present, the process is relatively straightforward, and only implies the modification of the Root.plist file in the bundle, which uses XML and can be modified with JDOM to include the required settings menu options.

However, when the project does not have a bundle already included, transformations to the project's pbxproj file are required to inject a custom bundle. This process was particularly difficult due to these files not having been designed for manual editing (and hence are quite difficult to understand for human readers), and not being documented by Apple online. The transformations required were mainly inferred from observation of several file samples from open-source iOS applications and non-official documentation found online. [46]

6.4 String ambiguity handling

As mentioned in the architecture portion of the report, the key disambiguation process consists of changing the resource keys so that they meet the following conditions:

- Two different keys do not have the same value.
- Modified values should have a number of characters as close to the original value as possible(preferably the same).

This process is necessary, due to the support libraries not having any effective means to differ between keys with the same value at runtime, making the assessment of the correct key to report unfeasible. This creates the need to make the key/value relationships univocal, with one value corresponding to exactly one key and vice versa. The size

preservation requirement is so that the UI is not distorted due to increased text length.

For this purpose, the ambiguous key values are modified during the project transformation process by replacing the final characters with an integer ID, incremented individually for each value. For example, if the value ”login” occurs on two different keys, the first occurrence will be replaced by ”logi1” and the second one by ”logi2”. The modified values are saved to the localized string files of the modified project.

The original and new key/value pairs for each application version are also stored in the SQLite database for coverage analysis and so that the tool can infer the original value after receiving association datasets.

Originally, the values were replaced fully by 'x' characters and each key was prevented from being reported more than once in the same context by dividing it into multiple ”subkeys” (one for each occurrence of the key in the source code). However, this approach was abandoned in the second semester after a usability test, as fully changing the string contents caused the UI to be overly complex to navigate. Prevention of repeated reports for the same key was implemented on the platform support libraries instead, as described in the next subsection.

6.5 Prevention of repeated string reports

During the process of detecting strings in the UI, it is desirable to prevent the support library from obtaining multiple screenshots for the same key in the same application context. However, it is highly desirable to obtain multiple screenshots for the key in different contexts, which is a common occurrence in real-world projects, as keys tend to be reused across the application extensively.

In order to prevent the string from being reported in the same context, the approach used originally consisted of dividing the keys with multiple occurrences in the source code into several subkeys and modifying the values like one would for two normal ambiguous values.

However, this approach was eventually abandoned, due to requiring the retrieval of key occurrences from the source code itself, implying extensive code analysis in the case of iOS projects that is otherwise

not needed.

Instead, both support libraries were modified to verify if the key has already been reported in the current application context before taking the screenshot and creating the association. Since both iOS and Android represent UI contexts using a tree-like structure, this was accomplished similarly in both libraries, by using hashes of the UI hierarchies. The process consists of:

- Whenever a key string is detected in the UI, a hash value is obtained from the string representation of the layout elements hierarchy of the current UI context (views, buttons, toolbar menus, etc), from the root to the leaves.
- A map is stored with the correspondence from the keys covered so far, and the list of hash values for the contexts they were detected in.
- Before taking the screenshot and associating with the key, the library checks if it has been reported for a context with the same hash value, if so the key is not reported it again. Otherwise, the hash value to the key's hash list in the map and the key/value/screenshot association is created normally.

While the approach does lead to very occasional coverage errors (i.e. a key being erroneously assessed as already reported for the given context) due to hash collisions, this occurrence will be rare, due to the efficiency of most string hash implementations for popular programming languages at preventing collisions.

7 Testing and evaluation

This section contains the tests performed on the tool over the course of the internship. It is also indicated which bugs were not fixed due to time constraints in the internship or other factors.

7.1 Unit tests

This section details the unit tests performed on each module of the tool. The legend for the test results is the following:

- Green: The expected behavior was achieved in all instances tested.
- Yellow: The expected behavior was achieved in the instances tested, with some limitations.
- Red: The expected behavior was not achieved to an acceptable level.

7.1.1 Web module

This section described the unit tests performed on the web module. The tests were performed on the web interface and HTTP server together. It should be noted that tests for this module related to uploading or transforming a project are only concerned with whether it has been uploaded successfully or whether it has been successfully retrieved after the transformation process occurred. Tests for the code transformation process itself are unrelated to these tests.

Test	Expected result	Limitations
Upload a project for transformation	The project is received successfully on the server.	N/A
Request transformation of the uploaded project	The transformed project is returned, or an error message is shown on screen if the transformation process fails.	N/A
Get list of subprojects and targets for the previously uploaded Xcode project	The subprojects are displayed correctly, and when one is selected list the targets obtained for them.	N/A

Show list for a particular application, platform and version	The datasets are displayed correctly	N/A
Display dataset details	The key/values present in the application are shown. The button ”View screenshots” appears next to the datasets for which screenshots have been obtained	N/A
Obtain the screenshots associated with a given key, in a given dataset	The screenshots are displayed correctly	N/A
Export a dataset to i18n in ”add new” mode	All the screenshots present in the dataset are exported to the platform, to their corresponding keys.	N/A
Export a dataset to i18n in ”delete all” mode	All the screenshots present in the dataset are exported to the platform and associated with the corresponding key. Previously added screenshots have been deleted from every key.	N/A
Export a dataset to i18n in ”only without” mode	The screenshots present in the dataset are exported to the platform, but only to keys without a screenshot attached already.	N/A

Table 5: Web module functional tests.

7.1.2 Transform module

This section details the unit tests performed on the operations performed by the transform module.

Test	Expected result	Limitations
Attempt to transform an uploaded Android project, with an incorrect relative path specified where there is no Android manifest	The project is deleted from the server folder and an error message is sent to the user.	N/A
Retrieve Android application's name from the Android manifest file	The name obtained is package name's last qualified name (after the last ".").	N/A
Retrieve the application's version name from the Gradle files, when there no product flavors declared (Android)	The version names declared on each product flavor are also obtained.	N/A
Retrieve the application's version name(s) from the Gradle files, when there are multiple versions declared in the product flavors (Android)	The version names declared on each product flavor are obtained, as well as the one in the default configuration.	N/A
Read the key/-values from the strings.xml file. The file does not have plurals or string arrays (Android)	The key/value pairs saved in the database are the correct key/values present in the file.	N/A

Read the key/-values from the strings.xml file. The file has multiple plurals and string arrays (Android)	The key/value pairs saved in the database are the correct key/values present in the file.	N/A
Disambiguate keys and save the modified versions (Android)	The modified keys are saved to the database and strings.xml file correctly.	N/A
Add the support library's aar file as a dependency in the Gradle file (Android)	After the build, the library can be accessed correctly from the source code.	N/A
Add the import statements (Android)	The import statements are added correctly to the source code.	N/A
Add "onMenuOpened" and "onPostResume" to Activity subclasse declarations, when the methods are overriden (Android)	The code snippets are added to the end of the methods and the project compiles successfully.	N/A
Add "onMenuOpened" and "onPostResume" to Activity subclasse declarations, when the methods are not overriden (Android)	The custom method declarations are added to the class declaration and the project compiles successfully.	N/A

Replace wrapped class identifiers with their ”wrapper” version (Android)	The class identifiers present in the code are replaced successfully and the application compiles successfully.	N/A
Obtain the list of subprojects within the project (iOS)	A list of xcodeproj file directories is returned, except for the ones inside the ”Pods” folder.	N/A
Obtain targets declared in a pbxproj file (iOS)	The correct list of targets shown when the project is opened is returned.	N/A
Attempt to transform an xcodeproj or target which does not exist within the project (iOS)	An error message is returned.	N/A
Attempt to perform transformations on an xcodeproj or target which does not exist within the project	An error message is returned.	N/A
Retrieve the application's short name and version from the Info.plist file	The name and version shown in the XML file are obtained.	N/A
Add export menu settings to the project's settings bundle, if present	The export menu is shown when the application's settings are accessed in the iPhone device or emulator.	N/A

Add settings bundle to the project when it does not have one (iOS)	The bundle is shown when the project is opened in Xcode, and the export menu is shown when the application's settings are accessed in the iPhone device or emulator.	N/A
Add the support library's pod to the Podfile, when present (iOS)	The pod install operation is successful, and the library can be accessed from the source code.	N/A
Add a Podfile to the project when not present, and add the library's pod to it afterwards (iOS)	The pod install operation is successful, and the library can be accessed from the source code.	N/A
Add the library import statement (iOS)	The library is initialized correctly during runtime, and the swizzling operations have been performed.	N/A

Table 6: Transform module tests.

7.1.3 Support libraries

This section describes the unit tests performed on the support libraries. It should be noted that tests performed on the use cases were done on small, sample applications which were dedicated to illustrating the usage of each UI structure. These tests were not concerned with coverage ratio, but rather to verify if the library was working properly for sample cases and without breaking the application at runtime.

Test	Expected result	Limitations
Read the application's name and version from application context and package manager	The name and version obtained are the same as the ones read during the transformation process.	Retrieving the name may fail when environment variables like $PRODUCT_NAME are used in naming.
Read the list of strings, string-arrays and plurals by performing reflection on the generated R class	The key/value pairs obtained are the same as during the transformation process.	N/A
Apply listener to a TextView	The string is detected in the view, when the view's textual contents change	N/A
Apply listener to a ViewPager	The strings are detected in the view and its children, when the page is changed	N/A
Apply listener to a TabLayout	The strings are detected in the view and its children, when the tab is changed	N/A
Apply listener to a ScrollView	The strings are detected in the view, when the device's screen is scrolled as they appear on screen	N/A
Create a Toast wrapper class instance with sample text, instantiate it and call "show"	The Toast is shown normally, and the string contained is detected. The screenshot taken displays the structure after it is fully rendered.	N/A

Create a Snackbar wrapper class instance with sample text, instantiate it and call ”show”	The Snackbar is shown normally, and the string contained is detected. The screenshot taken displays the structure after it is fully rendered.	N/A
Create an AlertDialog wrapper class instance with several inner buttons and textfields and call ”show”	The dialog is shown normally, and the strings contained are detected. The screenshot taken displays the structure after it is fully rendered.	N/A
Take a screenshot for a list of key/-values	The screenshot is saved to the filesystem, and the association is added to an in-memory HashMap correctly	N/A
Save the associations to the device’s filesystem	The associations are shown correctly in the JSON file in the folder	N/A
Zip and upload the folder with the associations, with a given dataset name. HTTP server is online	An error message is shown in the Android Studio logs	N/A
Trigger the export event with a given dataset name. HTTP server is online. Dataset with given name already exists for the current application/version	An error message is shown in the Android Studio logs	N/A

Trigger the export event with a given dataset name. HTTP server is offline. Dataset with given name already exists for the current application/version	An error message is shown in the Android Studio logs	N/A
Check the list of keys present in a view, when the view is not, or only partially, on screen. View is not a ViewGroup	The string contents are ignored	N/A
Obtain keys present in a View hierarchy (ViewGroup)	The view is traversed through for keys, if it is at least partially visible.	N/A

Table 7: Android library tests.

Test	Expected result	Limitations
Read the application's name and version	The name and version obtained are the same as the ones read during the transformation process.	N/A
Read the list of strings from the Localizable strings file	The key/value pairs obtained are the same as during the transformation process.	N/A
Swizzle a method from a given class	The new implementation for the method is used instead.	N/A

Take a screenshot for a list of key/-values	The screenshot is saved to the filesystem, and the association is added to an in-memory map correctly	N/A
Save the associations to the device's filesystem	The associations are shown correctly in the JSON file in the folder	N/A
Trigger the export event with a given dataset name. HTTP server is online.	The folder is zipped and sent to the server. The dataset is received successfully by the server.	N/A
Trigger the export event with a given dataset name. HTTP server is online. Dataset with given name already exists for the current application/version	An error message is shown in the Xcode logs	N/A
Trigger the export event with a given dataset name. HTTP server is offline. Dataset with given name already exists for the current application/version	An error message is shown in the Xcode logs	N/A
Get the list of keys present in a view, when the view is currently "tracked" and fully visible	Return the text present in the view, and all of its children.	N/A

Get the list of keys present in a view, when the view is not, or only partially, on screen and is currently tracked.	The view's own string contents are ignored, but its children are still checked for keys recursively.	When a view is partially obstructed by another, the keys will erroneously be detected as visible.
Get the list of keys present in a view, when the view is not currently "tracked"	The view's textual contents are ignored.	

Table 8: iOS library tests.

7.2 Integration tests

Integration tests were performed on the tool to make sure the complete system was working properly. In order to achieve this, the web module was hosted on a cloud-based emulated machine and the following process was performed for two Android and two iOS sample applications:

1. Transform the project in the web interface
2. Traverse the UI of the modified application, obtaining a partial list of the associations.
3. Upload the associations to the server as a dataset.
4. View the dataset in the web interface, checking if the obtained screenshots are shown properly.
5. Upload the dataset to i18n.

Whenever unforeseen errors were detected during the process, they were fixed. The applications used for the test are the same as the ones mentioned in the String coverage tests section.

7.3 Usability tests

During the second semester, two usability tests were performed on sample WIT-Software projects. The following improvements were

made as a result:

- Ambiguity handling was modified to make it less intrusive, as described in the Code transformation section.
- When transforming iOS projects, the user may select the xcodeproj file and the target to be transformed. The web interface lists them in a drop-down list before the transformation process.
- The transformation process was too slow due to the need to parse the entire project's source files with ANTLR. A heuristic was added to prevent the parsing of the files when not necessary on iOS projects, which consists of skipping the files which do not contain the "main" key word when searching for Objective-C files which contain a "main" function declaration declaration.

7.4 String coverage tests

In order to assess the tool's efficacy at obtaining the associations, coverage tests were performed on sample Android and iOS applications projects. The tests consisted of transforming the projects and executing the modified applications while keeping track of the ratio of the number of associations obtained by the number of associations tested (that is, the number of keys that actually appeared on the screen during the tests). False positives where keys were reported even though they were not present on screen were also tracked and analyzed.

For each platform, two applications were selected for this assessment, one representing a small-scale, proof of concept application to test basic use cases, and a larger, real-world application that more accurately represents the tool's performance at fulfilling its purpose in real scenarios. The results obtained are listed in Table 9.

Application	TODO-MVP [21]	Leafpic [23]	News-App [20]	Wikipedia-iOS [43]
Type	proof of concept	real-world	proof of concept	real-world
Platform	Android	Android	iOS	iOS
Keys tested	22	161	15	100
Keys obtained	20	155	15	98
Coverage ratio	90%	96%	100%	98%
False positives	0	2	0	8

Table 9: Key coverage on sample apps.

Analysis The coverage results for both platforms is acceptable, although the number of false positives is somewhat high. In Android, this is due to the tools menus, which are a difficult use case to handle properly.

The false positives in iOS were mainly due to partially obstructed views, which are common in scrollable menus. Assessing visibility of the text correctly in these cases is unfeasible, as it would require the library to check for overlaps in all the view combinations of each application context, leading to performance issues. This was deemed as an acceptable limitation of the tool, however.

8 Conclusion

This section assesses the work done across the internship and the results obtained, as well as future work that can be added to the project in the future. It also contains a personal note by the author about his experience during the internship.

8.1 Work done

At the beginning of the internship, the author was given a degree of flexibility in the scope of implemented features due to the uncertain nature of the project and his lack of experience. The priorities were the implementation of the Android modules and the export module, with the implementation of the iOS and web modules being dependant on the time remaining after finishing work on Android.

At the end of the internship, the implemented features include the Android and iOS transform modules and support libraries, the web module (back-end and front-end) and the export module for i18n. In addition, partial support is present for iOS projects implemented in Swift, only requiring the user to create the bridging header and importing the library. After many of the weekly meetings, various new features were proposed over the course of the second semester, including support for datasets, more robust library injection in the projects and integration with the WIT cloud. Most of them were implemented in time.

One predicted task that was notably not completed was the demonstration of the tool's functionalities in a WIT-Software project. This was due to time constraints and unforeseen mishaps by the author, which prevented a proper demonstration from being completed. The tool's features were nevertheless demonstrated on real-world applications.

In conclusion, the author considers that the project's overall results were positive. That being said, improvements can be made to existing features, as described in the future work section.

8.2 Future work

This section details additional features which can be added to the tool in the future, as well as possible improvements to existing features.

8.2.1 Web module

The web module's web interface was not the focal point of the internship, and therefore its design and functionality still have room for improvements. Depending on the changes made, it may be justifiable to use a more robust JavaScript library, such as React or AngularJS.

On the server side, adding sessions and user accounts would be a consideration, along with permissions to access datasets for each project.

8.2.2 Transformation module and support libraries

The transformation module was designed to be extensible to other platforms. Some considerations would include web (HTML/JavaScript) and the Unity game engine.

The module may be extended to give support to more programming languages in the already supported platforms, such as Swift on iOS and Kotlin on Android. Additional use cases may be handled, such as text drawn with Canvas in Android.

In the case of iOS specifically, the transformations performed to pbxproj files were particularly complex, and depend on the version of Xcode the project was built on. This section could be improved with further exploration to make sure it works properly on other versions.

The support libraries could be modified so that the position of the key in the screenshot is highlighted, for example by circling the text field with a colored rectangle before capturing the screenshot. This feature would make the screenshots more user-friendly to the translation team.

8.3 Final thoughts

This section will describe the author's thoughts on the internship. As it is mainly a personal subject, it is presented in the first person and in a somewhat informal tone.

First of all, from a technical standpoint, this internship has been overwhelmingly productive to me. I came into content to a very large set of technologies and development patterns, some of which I had very limited or no experience with before the internship began. This very broad set of IT topics and subfields includes:

- Android and iOS development.
- Static code analysis and transformation.
- Compilers, parsers and regular expressions.
- RESTful web services.
- Localization and Internationalization approaches and practices.
- Various other technologies, such as XML, JSON, SQLite, JavaScript and others.

I have also been involved in a real-world project that makes use of agile project management and planning practices, which, as an entry level worker in this field, provides invaluable working experience.

On a more personal level, I have thoroughly enjoyed the project. My tutor, supervisor and everyone else at WIT Software, along with my university supervisor, have proven to be extremely helpful and available to provide feedback when needed, and have done their very best to make the internship as a whole a positive experience to me, and the project itself has also been very interesting and enjoyable.

Appendices

A Internship context

This internship took place at WIT Software, S.A. (WIT), "a software company that creates advanced solutions and white-label products for the mobile telecommunications industry" [51], in the context of the author's Master's Degree in Software Engineering at the University of Coimbra.

Project Manager João Certo from WIT and Professor Ernesto Costa from the Department of Informatics Engineering at University of Coimbra supervised the internship. Lead Software Engineer Carlos Mota from WIT acted as the tutor.

The present document reflects the work done by the intern/author during the internship.

B Tests performed on the code analysis and transformation tools

In this section, a simple viability test was performed on some of the tools described in the state of the art, in order to evaluate the feasibility of using them in the present use case.

B.1 Test performed

The test consisted of performing small analysis and transformation tasks on a sample android studio application Java source file.
The idea of the test will be to change a file declaring an activity subclass to report the window keys when the activity either resumes or starts. This can be accomplished by adding the code fragment to the ”onPostResume” override method, before the final bracket, as shown in Figure 19.

```
//INJECTED CODE
LocalizationHelperAndroid.getInstance(this).reportWindowKeys();
```

Figure 19: onPostResume method call added.

If the method is not implemented in the given class, it needs to be implemented first. In that case, the following code snippet is added at the end of the class, before the final bracket (Figure 20):

```
//INJECTED CODE
@Override
protected void onPostResume() {
    super.onPostResume();
    LocalizationHelperAndroid.getInstance(this).reportWindowKeys();
}
```

Figure 20: onPostResume method snippet added.

While this is a relatively simple example, the transformation actually solves a lot of the use cases covered in Android by itself, as after the call to ”reportWindowKeys”, the localization library will implement listeners to do most of the work. [2]

B.2 ANTLR procedures

Note: For this test, the ANTLR Eclipse plugin was used. [9] The ANTLR GitHub repository contains ANTLR definitions (.g4 files) for the grammars of several languages, including Java. [10]

ANTLR (Python) The same test was performed for the ANTLR Python API, since Python tends to be a less verbose language than Java, and the author has a personal preference for the language. The API has much the same features as Java, and the implementation procedures were quite similar. The author found, however, that the Python API has a considerably less thorough documentation, and code examples tend to be harder to find online. Therefore, this option ended up being discarded.

Steps taken

1. The Java ANTLR grammar was obtained from the ANTLR grammar repository. Upon saving it on the .g4 file, the full Java code necessary for the parser was generated by the eclipse plugin as well as the abstract class "JavaBaseListener".

2. The class "ActivityResumeListener" was created. This class extends "JavaBaseListener" and overrides the method "enterClassDeclaration" which is called whenever the "TreeWalker" enters a Java class declaration (defined in the grammar file).

```
@Override
public void enterClassDeclaration(JavaParser.ClassDeclarationContext ctx) {

    //not an activity declaration
    if(!isActivity(ctx))
        return;

    //first we check if method already exists by walking the subtree
    OnPostResumeListener oprl = new OnPostResumeListener(ctx.depth() + 4);
    ParseTreeWalker.DEFAULT.walk(oprl, ctx);

    //if the method was found and changed, we have nothing else to do
    if(oprl.found())
        return;

    //otherwise, we create the onPostResume method before the last token (bracket in this case)
    JavaRunner.rewriter.insertBefore(ctx.getStop(), CODE_IF_METHOD_NOT_EXISTS);

}
```

Figure 21: Handling a class declaration node.

3. Firstly, one checks if the class extends either an "Activity" or an "AppCompatActivity". In order to accomplish this, the first child node after "extends" is traversed and its text contents are check. If the "extends" node does not exist it is also not an activity subclass.

```
//return true if rule is an activity declaration
private boolean isActivity(JavaParser.ClassDeclarationContext ctx) {
    String parentClass = null;
    //search for extends child. we know that the last child is the class body,
    //therefore we can stop searching before it
    for(int i = 0; i < ctx.getChildCount() - 1; i++) {
        if(ctx.getChild(i).getText().equals("extends")) {
            //parent class is the next child
            parentClass = ctx.getChild(i + 1).getText();
            break;
        }
    }

    //doesn't extend anything
    if(parentClass == null)
        return false;

    for(String name: ACTIVITY_TYPES) {
        //is a valid activity subclass name
        if(name.equals(parentClass))
            return true;
    }

    return false;
}
```

Figure 22: Checking if a class declaration extends an activity.

4. The next step consists of searching for the ”onPostResume” method declaration. In order to do this, another listener (”OnPostResumeListener”) is used to walk the class declaration subtree to search for methods with the ”Identifier” child node equal to ”onPostResume”.

   This listener ignores any method declaration with depth above 4 plus the parent class declaration node, in order to skip callback function declarations inside the class method’s statements. When it finds a method call ”onPostResume”, it adds the code snippet to the ending, and sets an inner attribute to true.

```
//auxiliary listeners
private static class OnPostResumeListener extends JavaBaseListener{

    //we only consider method bodies up to the max depth
    private int maxDepth;

    //will be set to true if we find onpostresume
    private boolean foundMethod;

    OnPostResumeListener( int maxDepth){
        super();
        this.foundMethod = false;
        this.maxDepth = maxDepth;
    }

    @Override
    public void enterMethodDeclaration(JavaParser.MethodDeclarationContext ctx) {
        //ignore this method declaration
        if(ctx.depth() > maxDepth)
            return;

        //check if method name is onPostResume
        if(ctx.getChild(1).getText().equals("onPostResume")) {
            foundMethod = true;

            //insert code before the last bracket
            JavaRunner.rewriter.insertBefore(ctx.getStop(), CODE_IF_METHOD_EXISTS);
        }
    }

    public boolean found() {
        return foundMethod;
    }
```

Figure 23: Listener to search for method ”onPostResume”

5. The ”parent” listener (”ActivityResumeListener”) will check if the method was found in the class body via that attribute, and simply return if it does. Otherwise, it adds the full method at the end of the class declaration.

B.3 CodeWorker procedures

The CodeWorker C++ source files were compiled in a Linux Environment. From that point, the test was performed using the CodeWorker interpreter, by running a CW script that took as arguments the directory of the input and output files. This could also have been accomplished done via the CW C++ API. [48]

A CodeWorker transformation rule was applied to the source file. This operates by simply parsing the file using a simple grammar, and making changes when certain patterns are found.

Steps taken by the rule

1. Start by creating auxiliary grammar rules. Define one rule for an activity class header, one for generic blocks of code placed between brackets and one for the ”onPostResume” method header.

```
//auxiliary grammars

//header for an activity subclass declaration
activity_header ::=
[
    "public"
    "class"
    #readIdentifier
    "extends"
    activity_subclass
]
;

//onPostResume method header
on_post_resume_header ::=

    "@Override" "protected" "void" "onPostResume" "(" ")"
;

//generic code fragment
misc_code ::=
    block
    |
    #readCString
    |
    [^'}']
;


//generic block statement
block ::=
    '{'
    //skip strings and inner blocks
    [#readCString | block | [^'}']]*
    '}'
;


//what we recognize as a valid activity subclass
activity_subclass ::=
    ["Activity" | "AppCompatActivity"]
;
```

Figure 24: CodeWorker auxiliary grammars.

2. Define a grammar rule for the case where the activity has the ”onPostResume” method defined. The rule is shown below.

```
with_method ::=
    #ignore(JAVA)
    activity_header
    "{"
        //skip rest of code on class body
        [~[on_post_resume_header | '}']]*
        [
            on_post_resume_header

            "{"
            //skip the rest of the method's code
            [misc_code]*
            //we insert the code snippet right before the last method bracket
            =>{
                //INJECTED CODE
                @
        LocalizationHelperAndroid.getInstance(this).reportWindowKeys();@
            }

            "}"
        ]
        //skip rest of code on class body
        [~[on_post_resume_header | '}']]*

    "}"

;
```

Figure 25: CodeWorker grammar for class with onPostResume.

3. Define the rule for the case where the class does not have the method implemented (meaning that it the transformation must insert it at the end of the class).

```
without_method ::=
        #ignore(JAVA)

        activity_header
        "{"

        [
            //only use this rule if the method onPostResume doesn't exist
            ~[on_post_resume_header | '}']
        ]*

        //inject code snippet just before the last bracket close
        =>{
        @
        //INJECTED CODE
        \@Override
        protected void onPostResume(){
            super.onPostResume();
            LocalizationHelperAndroid.getInstance(this).reportWindowKeys();
        }@
              }

       '}'
;
```

Figure 26: CodeWorker grammar for class without onPostResume.

4. Define the main rule, which recognizes both of the previous cases, and also skips the rest of the source file (imports, etc).

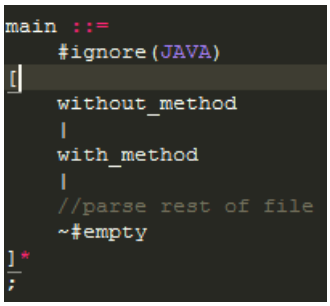


Figure 27: CodeWorker main grammar rule.

Test analysis While both tools accomplish the given task, they both appear to have their own merits. While CodeWorker makes it easier to handle the transformations themselves at specific portions of the code, it also requires one to define the necessary grammars for the task. ANTLR also has this requirement, but since its use is much more widespread, and the ANTLR GitHub repository itself already contains grammars for most popular programming languages (as well as many that are less popular), it will often not be necessary to create the grammars manually. It does, however, take time to study the structure of the grammars in order to perform more complex tasks. Furthermore, CodeWorker seems to be more error-prone when there is a need to handle more complex transformations, as one essentially needs to take into account every possible scenario when handling traversing the source code. For example, strings need to be skipped over to avoid the possibility of the code searched fragment being present inside the string.

ANTLR, on the other hand, already has the grammars defined for most languages, and one needs only to process the event of reaching the desired language structure required (expressions, statements,...). It also allows for a more trivial sense of context, as parse tree traversing allows the pinpoint of the current class, method and or/statement context, and go back and forth in it as necessary. The main drawback is that ANTLR was not designed for code transformations, and the process of inserting code snippets requires one to insert plaintext into a parse tree that is otherwise comprised of well defined tokens, which is less than ideal, but does solve the problem.

In the end, ANTLR (Java) was selected purely due to more thorough and easily accessible documentation (noticed during the tests), and the

author being more acquainted with the Java programming language, compared to the CodeWorker DSL.

C Implementation details

This section details some of the implementation details of the code transformation module, which makes use of the ANTLR Java API.

ParseTreeWalker For each source file obtained by the generic subcomponent, a "ParseTree" object is obtained, using the ANTLR grammar defined for that particular platform. The "ParseTreeWalker" class traverses the source code's parse tree, obtained with ANTLR. It takes a Listener parameter, which will be applied to the given Parse Tree. It is applied iteratively to each listener defined for the platform.

ANTLR listeners
ANTLR listeners trigger when a given ANTLR grammar rule occurrence has been found (such as expressions, statements and method calls) in the given parse tree. Inside the listener method definitions, the tree can be traversed freely. By traversing the tree to the current node's parent, child or sibling nodes, information about the the context where the listener was triggered can be inferred. This allows one to obtain more contextual information about the point in the code where the listener was triggered, as needed for the particular transformation case. For example, one can pinpoint the name of the method called in a method call, and the parameters used in it. It is also possible to obtain the code present in the node in plaintext format, also allowing for lexical analysis (using regular expressions).

The listeners allow for the detection of source code that follows specific syntactic and lexical patterns, in order to pinpoint parts that need to be modified in order to accomplish the necessary calls to the platform-specific library.

TokenStreamRewriter
The ANTLR "TokenStreamRewriter" class was used to perform the actual code transformations. This class injects the code fragments in plaintext before, instead or after a given ANTLR grammar definition. It is called inside the listeners, as needed to perform the required transformation for a particular use case in the current platform.
The ANTLR portion of Appendix B illustrates this implementation pattern, complete with code samples.

C.1 Android use cases

This section lists the use cases covered by the Android support library and describes the approach used for each one. The use cases were explored by modifying existing Android applications manually in the first semester of the internship and then later automatized through code transformations.

General case A method ("getVisibleKeysFromView") was implemented in the library, which obtains the total visible keys within a view, operating on its children recursively if needed (i.e. generic ViewGroup instances). Whenever a text field or other view that may contain text is found, all the keys contained within are reported. Non-visible views will be ignored. This method is always called using an asynchronous task, to prevent extra overhead to the UI thread and affect the responsiveness of the application. the "post" method of the view is also used, to ensure that it is only called when it is already constructed.

Before reporting any possible keys of the view, or its children, a check is performed on the view's on-screen visibility. For this purpose, the getVisibility() and getShown() methods often report false positives, so it is also checked if the view's rectangular area is being displayed on the screen currently.

```
//obtains key values visible inside a given mView group or view, and its child elements
//it also takes the Rect of the decorview as parameter, in order to check for visibility
private HashSet<String> getVisibleKeysFromView(final View v, final Rect decorRect) {
    HashSet<String> res = new HashSet<>();
    if (v == null)
        return res;

    //view not visible
    //if it's an instance of ViewStub, it is always invisible
    if (!(v.isShown() && v.getLocalVisibleRect(decorRect))) {
        return res;
    }
```

Figure 28: getVisibleKeysFromView signature and visibility checks.

Creating or resuming an activity When an activity subclass is entered, every key present in the root view of the current window is reported (including all child views). In order to accomplish this, at the end of the "onPostResume" method, the modified code will report every

key present in the current activity's window decor view. Initially, the method call simply used the activity's layout view. However, this is not a completely valid solution, as this view can be changed either in the "onCreate" method, or when the activity is resumed. Using "onPostResume" it is also ensured that the method is only called after the entire view tree is constructed (no null fields), as well as taking into account any changes made from the base layout.

```
@Override
protected void onPostResume() {
    super.onPostResume();

    /*
    KEY HERE: activity resumed execution
     */
    mLocalizationHelper.reportWindowKeys();
}
```

Figure 29: Code added to onPostResume.

Changing a TextView's textual contents Whenever the text contents of a text view change, the new values must be reported. In order to achieve this, a "TextWatcher" event listener was added to the view. This event listener will trigger every time the textual contents of the view change, and will report the new text fragments. [4]

ViewPagers and TabLayouts The "getVisibleKeysFromView" method, upon finding a "TabLayout" or "ViewPager" view, only reports the currently active child tab/page. In addition, to handle page changes, it uses the "addOnPageChangeListener"/"addOnTabSelectedListener" methods to add listeners that report the keys on the new tab/page.

```
//if mView is either a mView pager of tablayout, we need to figure out the currently selected page/tab
else if(v instanceof ViewPager){
    final ViewPager vp = (ViewPager) v;

    int selected = vp.getCurrentItem();


    if(selected >= 0);
        reportViewKeys(vp.getChildAt(selected));

    //add listener to handle page changes
    vp.addOnPageChangeListener(
            new ViewPager.SimpleOnPageChangeListener() {
                @Override
                public void onPageSelected(int position) {
                    LocalizationHelperAndroid.this.reportViewKeys(vp.getChildAt(position));
                }
            }
    );

}
```

Figure 30: Code fragment that handles ViewPagers.

Dialogs, Snackbars and Toasts These structures present a challenge, as they are not considered as views and accessing their internal components from the outside is not trivial. For example, AlertDialogs require one to use an "AlertDialogBuilder", initialize the components and proceed to build the actual dialog. As for Snackbars and Toasts, while it may seem that one only needs to check for the show() method call, it is not as trivial as it may seem. This is mainly due to the fact that one could instantiate the objects on completely unrelated methods, or even different files, before calling the show() method.

While tracking the variable reference would be possible by implementing a symbol table, this approach would be very difficult to implement and error-prone, while also requiring one to change the grammar files to support semantic actions, hindering their portability across languages and platforms.

Therefore, in order to handle these cases, "wrapper classes" were used. These classes will emulate the behavior of the original ones, with the exception that, when methods that show text on screen are called, the support library will be used to report the keys present in the given component's view hierarchy. The wrapper classes are included in the Android support library, and original class references are replaced by them using ANTLR listeners. [5] [3]

```
//if mView is either a mView pager of tablayout, we need to figure out the currently selected page/tab
else if(v instanceof ViewPager){
    final ViewPager vp = (ViewPager) v;

    int selected = vp.getCurrentItem();


    if(selected >= 0);
        reportViewKeys(vp.getChildAt(selected));

    //add listener to handle page changes
    vp.addOnPageChangeListener(
            new ViewPager.SimpleOnPageChangeListener() {
                @Override
                public void onPageSelected(int position) {
                    LocalizationHelperAndroid.this.reportViewKeys(vp.getChildAt(position));
                }
            }
    );

}
```

Figure 31: Example of a "wrapper" class definition for Snackbars.

Menus Menus presented a difficult challenge as well, for many reasons:

- Similar to Dialogs, Snackbars and Toasts they are not View subclasses.
- Their presentation views are usually displayed in a different Window instance, so they do not appear in the Activity's decoration view hierarchy. Therefore, simply taking a screenshot of the activity's decorview Bitmap will not display the menu components.
- Detecting when a menu finished its open/close animations is not trivial. If the animation has not finished when the screenshot is taken, the menu may not be shown correctly.

As a workaround to the issues mentioned the following procedures were used:

- A method call was added to the onMenuOpened method of each activity in the original application using ANTLR (method was generated if it was not present).
- The menu's textual components were obtained and reported in the aforementioned method call.
- A one-second sleep() call was used to wait for the menu's animation to finish. While this is not an optimal or universal solution, it serves nonetheless as a workaround for most cases.
- The MediaProjection class was used to take screenshots of the entire screen rather than just of the Activity's DecorView, in order to display menu components correctly in the screenshot.

C.2 iOS use cases

This section lists the use cases covered by the iOS support library and details the approach used for each one. The use cases were handled by performing swizzling operations on native iOS classes. It should be noted that the methods swizzled apply not only to the class itself, but also any classes extend from it. So, for example, calls to ”setText” on any view extending from UITextView will trigger the report process.

UIViewController In iOS, UIViewController objects are responsible for managing the view hierarchy, changing it according to user interaction and other events. They are mostly comparable to Android Activities, with the main difference being that in Android only one Activity can be active at a time, whereas in iOS it is very common to have multiple UIViewControllers active at the same time. Each controller is associated with a ”root” view in a hierarchy (which may itself be a part of another hierarchy, managed by a different controller) and is responsible for handling events related to that particular view as well as its children, including when the view appears and disappears from the screen. [15] Since it is only desirable to track UI changes on views with their view controller active, swizzling on the UIViewController class was directed at marking views as ”tracked” or ”untracked”. Untracked views are disregarded during the reporting process. The swizzling operations performed on the UIViewController class are detailed in Table 10.

Method name	Method description	Changes applied
viewDidAppear	Called after the root view associated with the view controller appears on the screen. [16]	When called, reports the controller's title string along with every visible view in the hierarchy. Also sets every view in it as ”tracked”.
viewWillDisappear	Called before the root view associated with the view controller disappears from the screen. [17]	When called, sets all the views in the hierarchy as ”untracked”.

Table 10: Swizzling for UIViewcontrollers.

UIView UIView is the class from which all the native interface element classes extend in iOS (similar to the View class in Android). The swizzling operations performed on this class are therefore meant to

handle method calls which require string reports regardless of the type of view object. The swizzled methods are detailed in Table 11:

Method name	Method description	Changes applied
didMoveToWindow	Called whenever a view moves to a new window. This can occur, for example, when the user scrolls the interface or the view's UIViewController is first created. [14]	When called, the view's own textual contents are reported if the view is a TextView or UILabel and is fully visible. Its children are recursively checked for visibility and reported if visible.
setHidden	Called when the view's visibility property changes from visible to invisible or from invisible to visible, either through a direct source code call or other means. This can occur, for example, when the user scrolls the interface or the view's UIViewController is first created.	When called, the view's own textual contents are reported if the view is a TextView or UILabel and is fully visible. Its children are recursively checked for visibility and reported if visible.

Table 11: Swizzling for UIViews.

It should be noted that in iOS every object that extends from UIView may have child views, unlike Android where only Views extending from the ViewGroup class contain them.

UITextView and UILabel These classes are used to represent text fields in iOS, similarly to the TextView class in Android [13] [12]. Only the "setText" method was swapped in these classes, as shown in Table 12.

Method name	Method description	Changes applied
setText	Called in the source code to change the textual contents of the view.	When called, the view's new textual contents are reported, if the view is fully visible.

Table 12: Swizzling for UITextViews.

D REST API documentation

This section describes the HTTP methods that make up the Web Module's REST API, including their parameters and responses.

POST /uploadassociations

Description:

Receives a dataset of key/value/screenshot associations from a modified application.

Parameters:

Name	Located In	Description	Required	Type
zipfile	body	Zip file containing the screenshots and a JSON file with the associations between the keys, their values and a list of the screenshots obtained for them.	YES	binary
dataset_name	body	Name chosen for the dataset	YES	string

Responses:

Code 204

Empty response

POST /submit

Description:

Uploads a project.

Parameters:

Name	Located In	Description	Required	Type
project	body	Zip file containing the screenshots and a JSON file with the associations	YES	binary
platform	body	The platform of the uploaded project	YES	string

Responses:

Code 204

```
Empty response
```

Code 200

```
Content: Xcodeproj paths/targets on each one, if the project is for iOS
Example:
{

    "/proj1/proj1.xcodeproj": [
        "targetA",
        "targetB"
    ],

    "/proj2/proj2.xcodeproj": [
        "targetA",
        "targetB"
    ]
}
```

POST /transform

Description:

Transforms the last uploaded project by the client IP and returns the modified version.

Parameters:

Name	Located In	Description	Required	Type
platform	body	The platform of the uploaded project	YES	string

Responses:

Code 201

Content: Zip file with the modified project

GET /getdatasets

Description:

Returns a list of created datasets for the given application, platform and version and the number of keys covered by it.

Parameters:

Name	Located In	Description	Required	Type
version	query	The name of the application version	YES	string
platform	query	The name of the platform	YES	string
application	query	The name of the application	YES	string

Responses:

Code 200

```
Content: The list of datasets and the number of
keys covered by each one
{
    "ds1": {
        "id": "1",
        "occurrences\_covered": "123"
    }
    ,

    "ds2": {
        "id": "2",
        "occurrences\_covered": "1234"
    }
}
```

GET /styles.css

Description:

Returns the CSS style sheet defined for the web interface.

Parameters:

None.

Responses:

Code 200

Content: The CSS style sheet defined for the web interface.

GET /index.html

Description:

Returns the home page HTML.

Parameters:

None.

Responses:

Code 200

Content: The home page HTML

GET /help.html

Description:

Returns the help page HTML.

Parameters:

None.

Responses:

Code 200

Content: The help page HTML

GET /transform.html

Description:

Returns the transform page HTML.

Parameters:

None.

Responses:

Code 200

Content: The transform page HTML

GET /about.html

Description:

Returns the about page HTML.

Parameters:

None.

Responses:

Code 200

Content: The about page HTML

GET /datasets.html

Description:

Returns the datasets web page HTML.

Parameters:

None.

Responses:

Code 200

Content: The datasets menu HTML page.

DELETE /deletedataset

Description:

Deletes the dataset with the given ID from the database and the file system.

Parameters:

Name	Located In	Description	Required	Type
datasetid	query	The id of the dataset	YES	integer

Responses:

Code 204

Empty response

GET /datasetdetails.html

Description:

Returns the dataset details web page for the dataset with the given ID.

Parameters:

Name	Located In	Description	Required	Type
datasetid	query	The ID of the desired dataset in the database.	YES	integer

Responses:

Code 200

Content: HTML of the dataset details page, for the given dataset

POST /logini18n

Description:

Logs into the i18n platform using the provided credentials and returns the list of i18n projects that the user has access to.

Parameters:

Name	Located In	Description	Required	Type
password	body	The user's i18n password	YES	string
username	body	The user's i18n username	YES	string

Responses:

Code 200

```
Content: The list of i18n projects that the user has access to
Example:
[
    "group1",
    "group2"
]
```

POST /exporti18n

Description:

Exports a dataset into the i18n platform to the given group, using the given export mode

Parameters:

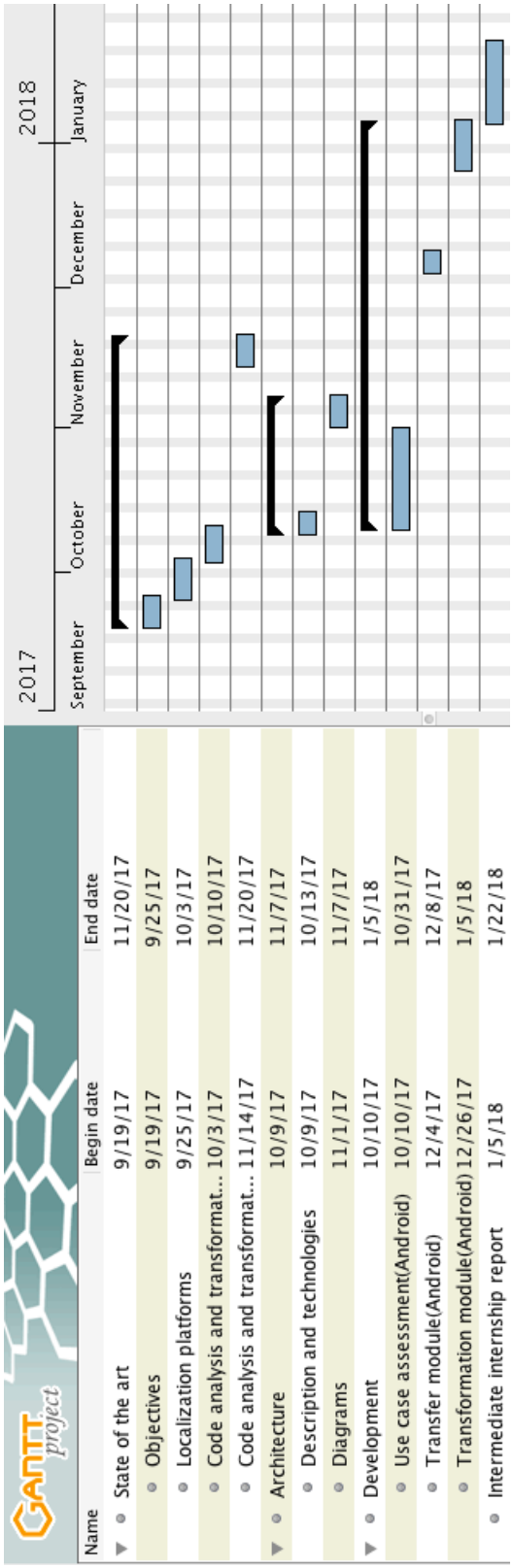
Name	Located In	Description	Required	Type
group	body	The i18n group to export the dataset to.	YES	string
mode	body	The export mode selected by the user(delete_all, add_new or only_without)	YES	string
datasetid	body	The dataset's ID in the database	YES	integer

Responses:

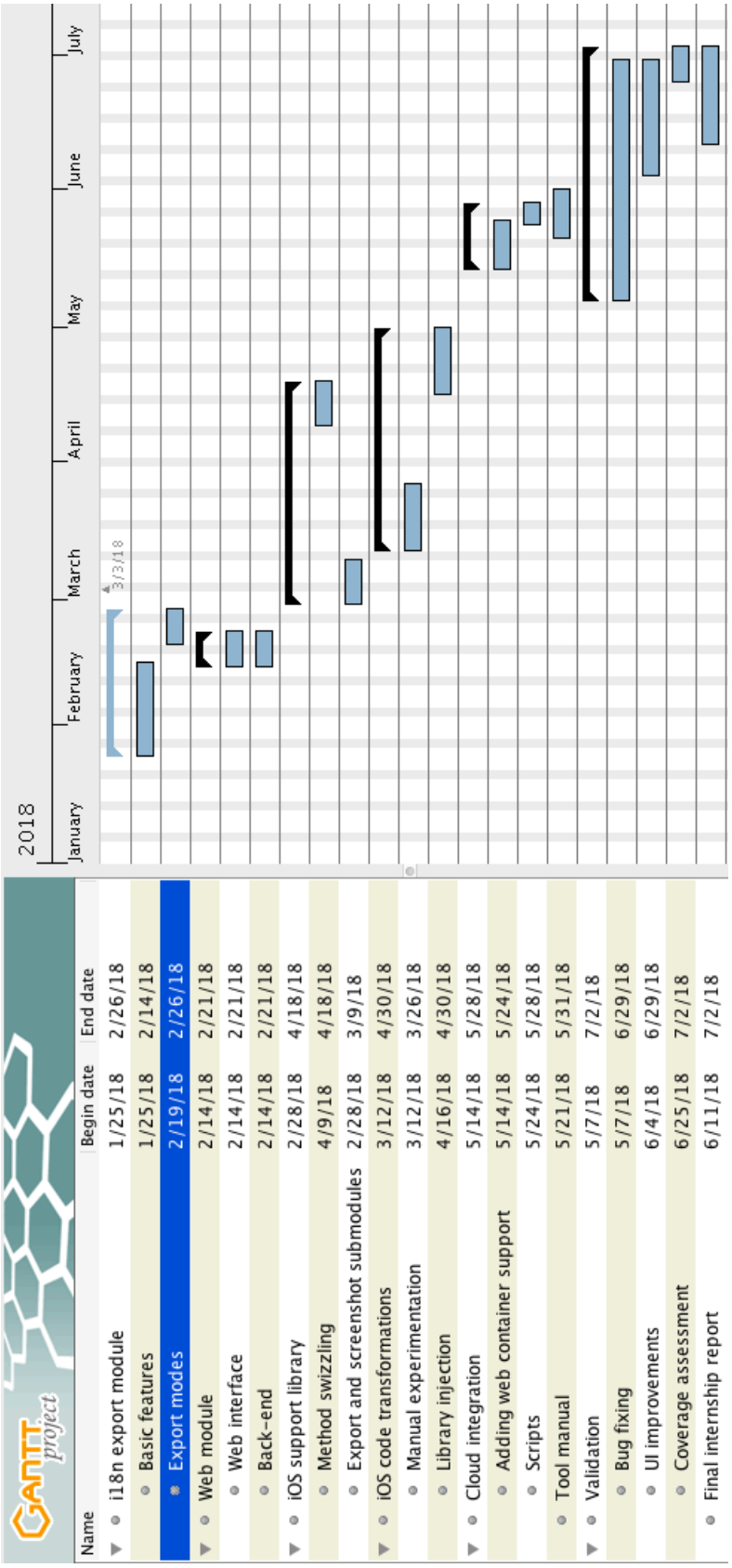
Code 204

Empty response

E First semester Gantt chart



F Second semester Gantt chart



G Weekly tasks

This section describes the work done on each week during the internship. The section was updated according to developments and conclusions on each week.

G.1 First semester

Week 1: 19/9/2017 - 20/9/2017

Developments and conclusions: The author received an introduction to the internship structure and objectives. He began researching the state of the art online.

Week 2: 20/9/2017 - 25/9/2017

Developments and conclusions: The author performed research about localization and localization support with his tutor and supervisor, for contextualization purposes. The results were documented.

Week 3: 25/9/2017 - 3/10/2017

Developments and conclusions: No tools were found that support the automation of screenshot association, although most could potentially be integrated with it, aside from Pootle. Results were documented.

Week 4: 3/10/2017 - 9/10/2017

Developments and conclusions: The tools FBInfer, ANTLR and Codeworker were analyzed summarily. FBInfer was deemed inappropriate for the project, due to not supporting code transformations out of the box, and the author not having experience with functional programming, required to modify the tool's source code.

Week 5: 9/10/2017 - 16/10/2017

Developments and conclusions: Some basic Android key reporting use cases were analyzed and handled manually, including Toasts, Snackbars and AlertDialogs. AlertDialogs were concluded to be a difficult use case to handle with code transformations. For this purpose,

a simple Android login template was modified to test the use cases (LoginActivity).

Week 6: 16/10/2017 - 23/10/2017

Developments and conclusions: A possible approach to report Notifications was explored. Screenshot and key/value association storage were implemented in the Android library. At this point, a more complex sample application was used (TODO MVP [21]), in order to test use cases in a more hands-on context. After attempting to detect keys in this application, a new one was chosen(Plaid [31]).

Week 7: 23/10/2017 - 31/10/2017

Developments and conclusions: Menus were detected as a particularly difficult use case to handle, due to not being a part of the normal View hierarchy. This week was mostly dedicated to preparing the first internship presentation at WIT Software.

Week 8: 1/11/2017 - 7/11/2017

Developments and conclusions: As it was concluded that the presentation slides could be improved, the author was given feedback and updated them accordingly this week, in preparation for the intermediate presentation at DEI.

Week 9: 7/11/2017 - 14/11/2017

Developments and conclusions: The architecture was defined for the overall system where the tool will be integrated. It was clarified that an intermediate web interface was necessary between the platform and i18n, as the platform may not have Internet access. The interface will be executed in the implementation machine(computer). A key ambiguity module was implemented, which modifies the strings.xml resource files in order to prevent value ambiguities.

Week 10: 14/11/2017 - 20/11/2017

Developments and conclusions: Some simple code transformation tasks were implemented using both ANTLR (Java) and CodeWorker.

The extensive test presented in Appendix A was performed on both tools. It was concluded that the ANTLR solution was more appropriate, due to being more extensible and maintained and the author's more extensive experience with the Java programming language.

Week 11: 20/11/2017 - 2/12/2017

Developments and conclusions: The same test was performed on the ANTLR Python API. It was concluded that the API did not have the same amount of support and documentation as the Java one, and did not offer many advantages over it and was therefore disregarded. Work was started on code transformations for real applications.

Week 12: 2/12/2017 - 10/12/2017

Developments and conclusions: Code transformations were implemented for the basic functionalities that could be supported by adding the "onPostResume" code fragment. The Android library's export module was implemented, along with the REST interface's method to receive the files. A simple debug web page was created to keep track of reported keys and their associated screenshots. The Java Spark micro web framework [22] was chosen due to its simplicity, and the project's use case not justifying the use of more powerful tools.

Week 13: 10/12/2017 - 18/12/2017

Developments and conclusions: AlertDialogs, Snackbar and Toast handling with code transformations was explored. Wrapper classes were implemented as a working solution, and the Code Transformation module was modified to replace class references.

Week 14: 18/12/2017 - 27/12/2017

Developments and conclusions: During the weekly meeting, it was concluded that keys that appear in the source code more than once can cause issues in coverage assessment, as they require the reporting of a given key more than once to achieve full coverage. A solution was implemented which divided these keys into multiple "subkeys" and replaced the key references across the source code. An issue was found

in the handling of menus, which prevented them from appearing in screenshots properly.

Week 15: 27/12/2017 - 2/1/2018

Developments and conclusions: A working solution was implemented for handling menus, using the "MediaProjection" library and screenshots of the whole screen.

Week 16: 2/1/2018 - 5/1/2018

Developments and conclusions: This week was spent testing the tool for multiple applications, and registering key coverage achieved so far.

Week 17: 5/1/2018 - 15/1/2018

Developments and conclusions: This week was dedicated to writing the intermediate report.

Week 18: 15/1/2018 - 22/1/2018

Developments and conclusions: This week was dedicated to finishing the intermediate report and reviewing it, along with the WIT tutor and supervisor, and the DEI supervisor.

Week 19: 22/1/2018 - 29/1/2018

Developments and conclusions: During this week, some final revisions were made to the intermediate report and work on the tool's export module for the i18n platform was started.

G.2 Second semester

Week 20: 12/2/2018 - 19/2/2018

Developments and conclusions: The i18n export module was finished and a basic web UI was created using HTML and JavaScript. Some of the revisions mentioned during the intermediate presentation were made to the internship report. Exploration of sample iOS

projects was also started.

Week 21: 19/2/2018 - 26/2/2018

Developments and conclusions: During the weekly meeting, it was decided that the i18n export module should have three export modes defined and that multiple association datasets should be stored for each application and platform (more details in the Architecture section of the report). The module was changed to incorporate the export modes and work was started on the persistence submodule to store and manage the datasets in an SQLite database.

Week 22: 26/2/2018 - 5/3/2018

Developments and conclusions: Work was finished on the persistence submodule. The iOS export module was started using the Alamofire library [1] in Swift.

Week 23: 5/3/2018 - 12/3/2018

Developments and conclusions: The export module was finished, along with the screenshot capture iOS submodule. Wrapper classes were studied as a possible approach for the implementation of the code transformations in iOS.

Week 24: 12/3/2018 - 19/3/2018

Developments and conclusions: The iOS settings bundle was defined as the entry point for the iOS user event that triggers the associations export action. The user action was implemented and more iOS use cases were analyzed.

Week 25: 19/3/2018 - 26/3/2018

Developments and conclusions: After a meeting with an iOS developer from WIT, it was concluded that method swizzling was a better approach for handling transformations in iOS, due to being less error-prone and covering more cases. The author began exploring the mechanism. During this week, the tool was also tested on a WIT

Android project. Bugs were reported and fixed and it was concluded that a better approach needed to be used to handle ambiguities, due to the current approach being too disruptive when traversing the UI.

Week 26: 2/4/2018 - 9/4/2018

Developments and conclusions: During the weekly meeting, approaches were discussed to handle key ambiguities in a less intrusive way. It was decided that value-based ambiguities would be solved by changing only the final characters in each string. After experimenting with various approaches, it was decided keys with multiple occurrences would be handled by checking if they were already reported in the current UI context, using a hash algorithm on the view hierarchy.

Week 27: 9/4/2018 - 16/4/2018

Developments and conclusions: Swizzling operations were experimented with in the Wikipedia iOS app. The library was converted to a CocoaPod for easier injection in the projects.

Week 28: 16/4/2018 - 23/4/2018

Developments and conclusions: The swizzling operations required to handle the explored use cases were defined. Podfile transformations were implemented by modifying a Ruby grammar from the ANTLR grammar repository. Work was started on the injection of a custom settings bundle package in the projects to handle the iOS user event.

Week 29: 23/4/2018 - 30/4/2018

Developments and conclusions: An ANTLR grammar was created to perform transformations on Xcode pbxproj files, in order to include the settings bundle package. The iOS transformation module was tested on other apps.

Week 30: 30/4/2018 - 7/5/2018

Developments and conclusions: During this week, the author

made some revisions on the internship report. Key coverage was accessed on the Wikipedia iOS application to evaluate progress at that point. It was concluded that only views which were partially obstructed by others were not being handled properly.

Week 31: 7/5/2018 - 14/5/2018

Developments and conclusions: During the meeting, it was concluded that the datasets should be associated with individual application versions on the same platform. The web module was changed to incorporate the changes, and the Gradle file code analysis was changed to be able to read versions across multiple product flavors.

Week 32: 14/5/2018 - 21/5/2018

Developments and conclusions: During the meeting, it was decided that the web module should be able to run in a web container (the tool only ran using the embedded Jetty server from Spark at this point) and on a WIT-Software cloud. Changes were made to the web module to incorporate the changes.

Week 33: 21/5/2018 - 28/5/2018

Developments and conclusions: The cloud was set up with the tool's dependencies. Various Python scripts were created to export the application in both standalone mode and in a container (tested on Wildfly). Minor usability fixes were done on the web interface. Work was started on the tool's configuration and usage manual.

Week 34: 28/5/2018 - 4/6/2018

Developments and conclusions: The help menu was added to the web interface. The manual was finished, including small changes to reflect interface updates. During the meeting, it was decided that the tool would be assessed and undergo changes with the assistance of Software Engineer Duarte Costa from WIT.

Week 35: 4/6/2018 - 11/6/2018

Developments and conclusions: During this week, small usability issues were fixed in the web interface, and the tool was tested on a WIT iOS project. Errors were found and fixed.in the Podfile transformations.

Week 36: 11/6/2018 - 18/6/2018

Developments and conclusions: During this week, it was decided that the tool should make it possible to list and select Xcode proj files and targets within the submitted project before the transformation process, which required the implementation of additional pbxproj code analysis steps. The week was also spent working on the internship report.

Week 37: 18/6/2018 - 25/6/2018

Developments and conclusions: This week was mostly spent working on the report, with the additional fixing of some errors in the iOS transform module.

Week 37: 18/6/2018 - 25/6/2018

Developments and conclusions: This week was mostly spent working on the report.

References

[1] Alamofire - github repository. https://github.com/Alamofire/Alamofire.
Accessed: 24/06/2018.

[2] Android documentation - activity. https://developer.android.com/reference/android/app/Activity.html.
Accessed: 29/9/2017.

[3] Android documentation - snackbar. https://developer.android.com/reference/android/support/design/widget/Snackbar.html.
Accessed: 11/12/2017.

[4] Android documentation - textwatchers. https://developer.android.com/reference/android/text/TextWatcher.html.
Accessed: 17/11/2017.

[5] Android documentation - toast. https://developer.android.com/reference/android/widget/Toast.html.
Accessed: 11/12/2017.

[6] Android documentation - view. https://developer.android.com/reference/android/view/View.html.
Accessed: 3/10/2017.

[7] Android documentation - viewgroup. https://developer.android.com/reference/android/view/ViewGroup.html.
Accessed: 3/10/2017.

[8] Android multi-language app. https://www.androidhive.info/2014/07/android-building-multi-language-supported-app/.
Accessed: 28/3/2018.

[9] Antlr eclipse plugin repository and tutorial. https://github.com/antlr4ide/antlr4ide.
Accessed: 10/11/2017.

[10] Antlr grammars github repository. https://github.com/antlr/grammars-v4.
Accessed: 11/10/2017.

[11] Apple documentation - nsobject load() method. https://developer.apple.com/documentation/objectivec/nsobject/1418815-load?language=objc.
Accessed: 24/06/2018.

[12] Apple documentation - uilabel class. https://developer.apple.com/documentation/uikit/uilabel?changes=_2.
Accessed: 24/06/2018.

[13] Apple documentation - uitextview class. https://developer.apple.com/documentation/uikit/uitextview.
Accessed: 24/06/2018.

[14] Apple documentation - uiview didmovetowindow method. https://developer.apple.com/documentation/uikit/uiview/1622527-didmovetowindow.
Accessed: 24/06/2018.

[15] Apple documentation - uiviewcontroller class. https://developer.apple.com/documentation/uikit/uiviewcontroller.
Accessed: 24/06/2018.

[16] Apple documentation - uiviewcontroller viewdidappear method. https://developer.apple.com/documentation/uikit/uiviewcontroller/1621423-viewdidappear.
Accessed: 24/06/2018.

[17] Apple documentation - uiviewcontroller viewwilldisappear method. https://developer.apple.com/documentation/uikit/uiviewcontroller/1621485-viewwilldisappear.
Accessed: 24/06/2018.

[18] Cocoapods website. https://cocoapods.org/.
Accessed: 24/06/2018.

[19] Early Years of Unicode. https://www.unicode.org/history/earlyyears.html.
Accessed: 5/6/2018.

[20] Github - news application sample app for ios. https://github.com/yavuz/News-Application.
Accessed: 2/04/2018.

[21] googlesamples/android-architecture. https://github.com/googlesamples/android-architecture/tree/todo-mvp/.
Accessed: 3/10/2017.

[22] Java spark homepage. http://sparkjava.com/.
Accessed: 8/12/2017.

[23] Leafpic github repository. https://github.com/HoraApps/LeafPic.
Accessed: 27/12/2017.

[24] Localizing with resources. https://developer.android.com/guide/topics/resources/localization.html.
Accessed: 15/2/2018.

[25] Maroun bassam - must know: Unicode and character sets: Ascii, unicode and utf-8. http://marounbassam.blogspot.com/2016/02/must-know-unicode-and-character-sets.html?spref=tw.
Accessed: 19/2/2018.

[26] Microsoft developer network - about domain specific languages. https://msdn.microsoft.com/en-us/library/bb126278.aspx.
Accessed: 2/1/2018.

[27] Microsoft docs - units of measurement. https://docs.microsoft.com/en-us/globalization/locale/units-of-measurement.
Accessed: 1/2/2018.

[28] Multilizer - delphi and c++builder. http://help.multilizer.com/devzone/delphi-cbuilder/.
Accessed: 25/9/2017.

[29] Multilizer - features. http://www2.multilizer.com/features/.
Accessed: 22/9/2017.

[30] NShipster - Method Swizzling. http://nshipster.com/method-swizzling/.
Accessed: 5/6/2018.

[31] Plaid application repository. https://github.com/nickbutcher/plaid.
Accessed: 18/10/2017.

[32] Pootle documentation. http://docs.translatehouse.org/projects/pootle/en/stable-2.8.x/.
Accessed: 21/9/2017.

[33] Pootle features. http://docs.translatehouse.org/projects/pootle/en/stable-2.8.x/features/index.html.
Accessed: 21/9/2017.

[34] Quick starter on parser grammars - no past experience required. https://theantlrguy.atlassian.net/wiki/spaces/ANTLR3/pages/2687210/Quick+Starter+on+Parser+Grammars+-+No+Past+Experience+Required.
Accessed: 2/1/2018.

[35] A simple approach to access localized resources in windows store apps. https://marcominerva.wordpress.com/2013/02/09/a-simple-class-to-access-localized-resources-in-windows-store-a

Accessed: 16/2/2018.

[36] Stack Overflow post by user INgeek about UTF character encodings. https://stackoverflow.com/questions/2241348/what-is-unicode-utf-8-utf-16#answer-41708331.
Accessed: 5/6/2018.

[37] Text united - features. https://www.textunited.com/software-localization/.
Accessed: 25/9/2017.

[38] Transifex - api. https://docs.transifex.com/api/introduction.
Accessed: 25/9/2017.

[39] Transifex - features. https://www.transifex.com/features/.
Accessed: 25/9/2017.

[40] W3C - Localization vs Internationalization. https://www.w3.org/International/questions/qa-i18n.
Accessed: 17/06/2018.

[41] W3C - Script direction and languages. https://www.w3.org/International/questions/qa-scripts.
Accessed: 1/2/2018.

[42] What is a backlog? | agile alliance. https://www.agilealliance.org/glossary/backlog/#q=~(filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'backlog))~searchTerm~'~sort~false~sortDirection~'asc~page~1).
Accessed: 10/1/2017.

[43] Wikipedia ios app github. https://github.com/wikimedia/wikipedia-ios.
Accessed: 17/06/2018.

[44] Wichmann B.A., Canning A.A., Clutterbuck D.L., Winsborrow L.A., Ward N.J., and Marsh D.W.R. Industrial perspective on static analysis. Software Engineering Journal (Volume: 10, Issue: 2, pp 69 - 75, March 1995).
Accessed: 20/9/2017.

[45] Carrie Cousins. Color and cultural design considerations. https://www.webdesignerdepot.com/2012/06/color-and-cultural-design-considerations/.
Accessed: 1/2/2018.

[46] Laurent Etiemble. Monobjc - xcode project file format. http://www.monobjc.net/xcode-project-file-format.html.
Accessed: 2/05/2018.

[47] Cedric Lemaire. Codeworker discontinued. http://www.codeworker.org/download-codeworker.
Accessed: 22/10/2017.

[48] Cedric Lemaire. Codeworker tutorial. http://codeworker.free.fr/tutorials/DesignSpecificModeling/tutorial.html.
Accessed: 22/10/2017.

[49] Cedric Lemaire. A universal parsing tool & a source code generator. http://codeworker.free.fr/.
Accessed: 22/10/2017.

[50] Terrence Parr. About the antlr parser generator. http://www.antlr.org/about.html.
Accessed: 11/10/2017.

[51] WIT Software. Why wit.
https://www.wit-software.com/company/why-wit/, 2015.
Accessed: 13/11/2017.