UNIVERSITÄT BIELEFELD
Faculty of Technology

ZAB Zentrale Anlaufstelle Barrierefrei

UniMaps Bielefeld

Automatic Code Generation for a Cross-Platform Indoor Navigation App

Participants: Dorian Drost (ddrost@techfak.uni-bielefeld.de), Rebecca Fortman (rfortmann@techfak.uni-bielefeld.de), Christian Kullik (ckullik@techfak.uni-bielefeld.de), Dustin Matzel (dmatzel@techfak.uni-bielefeld.de), Bettina Reglin (breglin@techfak.uni-bielefeld.de)

Supervisors: Sebastian Wrede (sebastian.wrede@uni-bielefeld.de), Jan Moringen (jmoringe@techfak.uni-bielefeld.de), Michael Johannfunke(johannfunke@uni-bielefeld.de)

Abstract — Smartphones nowadays mainly run one of two operating systems, namely Android or iOS. When creating variants of an application for both systems, the application developer has to adopt the programming model and frameworks dictated by the respective system while the core functionality of the application remains the same between both variants. Within this project, we used crosscompilation from Java to Objective-C to reuse the backend of an existing Android application for creating an iOS version of the same.

UniMaps

UniMaps is an Android application for indoor navigation on the campus of the Bielefeld University. A first prototype was developed within the course „Software Engineering“ in early 2017 and is developed further since then by students from the technical faculty. The application is developed on behalf of the „Zentrale Anlaufstelle Barrierefrei“ of the Bielefeld University and is designed to especially consider the needs of handicapped and disabled users. Beyond navigating on the campus, UniMaps also provides useful information like the canteen menu, the schedule of the tram and the possibility to customize UniMaps according to personal needs.

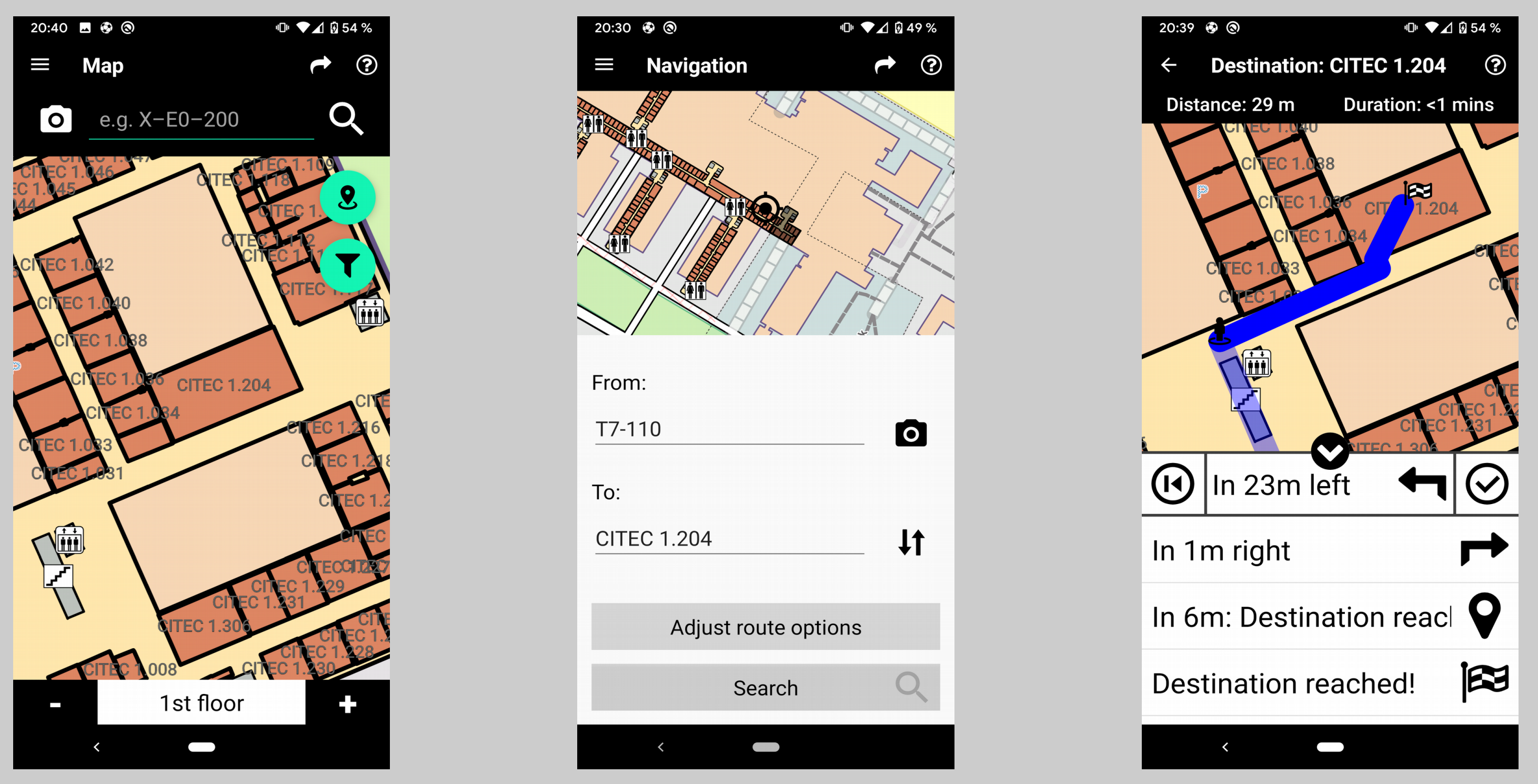


Figure 1: UniMaps main feature is navigating on the campus and displaying the map.

Apps for both systems

UniMaps was designed and developed for Android smartphones. However, to reach as many users as possible, an application for Apple's iOS system is required. While some parts of the application are platform specific (e.g. implementation of the database, User Interface), large parts of the code base are algorithm-centric and thus independent of the target platform (e.g. the A*-algorithm for the navigation, that is written in non-platform-specific Java code). In order to avoid unnecessary and redundant work, those parts should be reused by the iOS application.

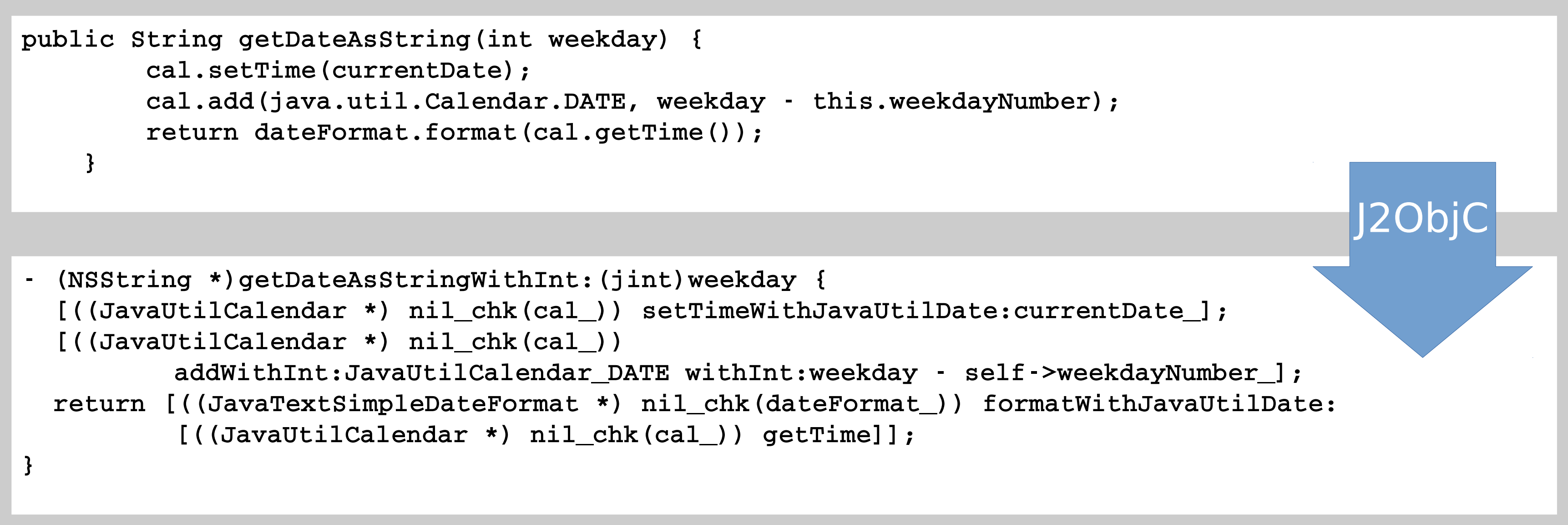
To display the map, we used Mapsforge[2]. Mapsforge is an open source Android library that is capable of converting osm data to a custom format, which takes up little memory and allows fast processing of the map data.

When faced with the problem of creating an application for both Android and iOS smartphones, two main approaches exist that save one from programming the app twice. Those are code generation and crosscompilation. The first one expects the code-basis to be written in a programming language with a higher level of abstraction to be translated to both Java and Objective-C code. Crosscompilation, on the other hand, transforms code from one language to another (in this case from Java to Objective-C).

As the Android application written in Java already exists, crosscompilation is the method of choice here.

Preparing the structure

A Java class one wants to crosscompile must not contain any android-specific code. In order to fulfill this requirement, the structure of the Android project had to be refactored largely. All platform-independent code was moved to a new module we call the backend. This backend provides the core functionality for both applications. In the Android project, it is imported as a Java library, while for the iOS project it acts as source for the crosscompilation.


```
public String getDateAsString(int weekday) {
        cal.setTime(currentDate);
        cal.add(java.util.Calendar.DATE, weekday - this.weekdayNumber);
        return dateFormat.format(cal.getTime());
    }
```

```
- (NSString *)getDateAsStringWithInt:(jint)weekday {
  [((JavaUtilCalendar *) nil_chk(cal_)) setTimeWithJavaUtilDate:currentDate_];
  [((JavaUtilCalendar *) nil_chk(cal_))
          addWithInt:JavaUtilCalendar_DATE withInt:weekday - self->weekdayNumber_];
  return [((JavaTextSimpleDateFormat *) nil_chk(dateFormat_)) formatWithJavaUtilDate:
          [((JavaUtilCalendar *) nil_chk(cal_)) getTime]];
}
```
Figure 2: A crosscompiler transforms code from one programming language into another.

Crosscompiling the backend

With the J2ObjC[1] crosscompiler, we created Objective-C code from a total of 7000 lines of Java code. This code contains the functionality for computing the shortest path for a route, parsing data from the canteen as well as the PEVZ API and implements the tram schedule.
The crosscompilation became part of the Android CI-pipeline, assuring that the Android backend is always crosscompileable. The iOS project imports and crosscompiles the code automatically, considering changes in the backend with every new build.

Figure 3: Download UniMaps in the Google Play Store

While we where able to sucessfully crosscompile large parts of the project, crosscompiling the Mapsforge library was not doable. In order to display the maps on the screen, we had to find a substitute for Mapsforge. Unfortunately no other library fulfilled our demands (i.e. being able to display custom maps and being compatible with the used licences).

Discussion

While having the potential to avoid reprogramming code that already exists in another programming language, crosscompiling also demands prerequisites. Fulfilling those requires a certain effort to be taken. Among those prerequisites is a structure of the code that separates frontend and backend in a very distinct way. The more code to crosscompile, the more this effort is worth to be taken, obviously.

In our case crosscompiling the backend rapidly reduced the effort of the iOS application. In particular, as within the team there was no expertise in the Objective-C language, reprogramming the backend for the iOS application would have required the team members to learn a new programming language, which is costly in time and resources. By using crosscompilation this was avoided.

Conclusion

We see that crosscompiling is a satisfying way to reuse non-platform-specific code for applications on different operating systems. With the approach we described above, the backend can still be maintained in the sameway as before (namely as a Java library) and changes directly apply to both applications.

We further saw, that the iOS project had huge positive influence on the Android project as well, as the crosscompilation forced a decent structure on the project's packages.

[1] https://github.com/google/j2objc [2] https://github.com/mapsforge/mapsforge