BuildApp: A web-based integrated development environment for Java, Android and a new programming language

Poon Stirling Yeu
in collaboration with Shum Chi Chung and Wong Man Chun

supervised by
Dr. TW Chim

April 17, 2016
Abstract

Despite the increasing popularity of using an IDE for application development, traditional IDE platforms impose software and hardware requirements on the user’s computer, setting up a barrier of entry for people unable to afford a performant computer. Furthermore, popular languages such as Java and C++ supported by these IDEs encourage bad coding habits and hinder developers’ productivity. Therefore, there is a need to develop a machine-independent cloud-based collaborative IDE which can reduce dependence on the user’s computer. Additionally, to enable higher developer productivity, a new programming language will be introduced and incorporated into the new IDE.
Acknowledgements

We would like to thank our teachers and advisors, Dr. T.W. Chim, Dr. C.K. Chui, Dr. Jon Hui and Prof. Bruno C. d. S. Oliveira for their advice and guidance.

This project uses materials provided by the following entities.

Copyright (c) 2010, 2012, 2013 The Go Authors. All rights reserved.
jQuery v1.11.3 (c) 2005, 2015 jQuery Foundation, Inc.
Copyright (c) 2013 Bootsniпп.com
Copyright (c) 2014 Ashley Jeffs
Copyright 2016 Microsoft Azure
Copyright (c) 2014 Cenk Alt
Copyright 2016 Gary Burd
Copyright (c) 2012 Rodrigo Moraes. All rights reserved.
Copyright 2012 The Go-MySQL-Driver Authors. All rights reserved.
Copyright (c) 2011-2013, “pq” Contributors Portions Copyright (C) 2011 Blake Mizerany
Copyright (c) 2009,2014 Google Inc. All rights reserved.
Copyright (c) 2014 Andy Smith
Copyright (c) 2010, Ajax.org B.V. All rights reserved.
Copyright Joyent, Inc. and other Node contributors. All rights reserved.
Copyright (c) 2014 Ivan Bozhanov
Copyright (c) 2011-2016 Twitter, Inc.
Copyright (c) 2012-2013, Christopher Jeffrey (https://github.com/chjj/)
Copyright (c) 2014-2015 Automattic (dev@cloudup.com)
Float Label Pattern Plugin for Bootstrap 3.1.0 by Travis Wilson
Copyright (c) 2014-2015 almasaeed2010
Copyright (c) 2015 Tony Cabaye
Copyright (c) 2013 Alex Lockwood
Copyright (c) 2014-2016 almasaeed2010
Copyright (c) 2010-2013 - Gustavo Niemeyer (gustavo@niemeyer.net)
Copyright [2015] Contributors: Jacek Marchwicki (jacek.marchwicki@gmail.com), Karol Wojtaszek (karol@appunite.com)
jacekmarchwicki/android (https://github.com/appunite/docker)
ksoichiro/android (https://github.com/ksoichiro/dockerfiles)
steinwaywhw/client.js (https://gist.github.com/steinwaywhw/9920493)

Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries. Docker, Inc. and other parties may also have trademark rights in other terms used herein.

Android is a trademark of Google Inc.
Contents

1 Introduction 6

2 Objectives and Benefits 8

3 Existing Work 11

4 Scope 12

5 Design and Implementation 13
  5.1 Platform setup ........................................... 13
  5.2 Web-based Integrated Development Environment (IDE) front-end ........................................... 13
  5.3 Web-based IDE back-end ................................. 15
    5.3.1 Use of Go language as implementation language ........................................... 15
    5.3.2 Database ........................................... 16
    5.3.3 Compiling and executing environment ........................................... 16
  5.4 Agent application ........................................ 16
  5.5 Security design ........................................... 16
  5.6 New programming language ............................. 17
    5.6.1 Selection of XText as framework ..................... 18
    5.6.2 Core components .................................... 21
    5.6.3 Integration with web IDE ............................ 25

6 Deliverables 27
  6.1 Web-based IDE front-end ............................... 27
  6.2 Web-based IDE back-end ............................... 27
  6.3 Agent application ....................................... 28
  6.4 New programming language ............................ 28

7 Division of Labour 29

8 Results and Challenges 30
9 Future plans
  9.1 Smart preview .................................................. 32
  9.2 Collaborative editing algorithm ............................... 33

10 Conclusion ......................................................... 36
Chapter 1

Introduction

IDEs are a key programming tool for many programmers, providing key components in an integrated interface. For complex projects that target mobile devices, an IDE that bundles all the tools needed along with predefined workflows that can build most applications are greatly beneficial for programmers to start programming out-of-the-box.

As of April 2016, Eclipse, Visual Studio and Android Studio are the 3 most popular IDE by search volume [6]. However, all three of these traditional IDEs lack native real-time team collaboration support, which Kats et al. [18] believes causes traditional IDEs to be inefficient “sharing pipelines” when developers need to work with each other. Furthermore, having to install these on each programmer’s machine can be time-consuming and error-prone [9]. This discourages beginners and teams that want to collaborate without having to set up everything from scratch.

To tackle these issues, Boehm et al. [4] proposes that productivity could increase if developers where given access to coding platforms that can be accessed easily. Several advantages of an online IDE makes it a natural choice. Since it is centralized, it can avoid installing anything locally apart from a browser, while having access to a common copy prevent version conflicts and allows effective sharing pipelines [18].

This project intends to innovate in three key areas. First, modern software engineering practices are incorporated into the IDE platform. For example, the issues module allows programmers to view users side-by-side with the coding panel. Second, this projects aims to reduce the development, testing and deployment time by creating the methods of directly pushing applications onto mobile devices and testing applications within siloed virtual machines. Third, a new programming language is introduced to foster development speed and to ease development complexity.

The remainder of the report will proceed as follows. The objectives and
benefits of this project are discussed, followed by the scope and the deliverables of the project. Then, it looks at design and implementation of the different project components. Finally, future work for this project would be addressed before closing up with a conclusion.
Chapter 2
Objectives and Benefits

This project aims to build a web-based IDE that incorporates code editing, project management, compilation and debugging tools for Android and Java development, and enhances developers’ productivity with a new translation language. These criteria for the IDE permit it to offer a more effective workbench that enables modern programming techniques seamlessly.

As discussed in the introduction, a cloud-based IDE platform is better suited to today’s “sharing pipelines” amongst programmers, with developers on the go coding anywhere with an Internet-connected computer, and can start building applications without having to install or configure anything on the local computer.

Android is chosen as the mobile platform that can be developed in the IDE because of the higher penetration rate of Android devices, higher transparency of source code and lower technical difficulty. Mobile application usage has experienced significant growth in recent years [19, 3], with a market share of over 80% in the mobile operating system market [17] in the second quarter of 2015. Due to the open nature of Android [16], it is easier to push applications developed by the IDE to Android devices by installing APK files [16, 13].

With the rise of mobile development brings the problem of having to test the application on a different variety of common phones and tablets. This usually entails having to plug in each device into the computer and deploy the application one-by-one. Our approach obsoletes this cumbersome practise where the IDE can, cable-free, push the application to multiple devices upon a single click. The developer also does not need to check each device for the test results; the IDE will display the test results of each device inside the IDE. For measurable and quantifiable testing process, the developer can opt for the screen recordings of the devices to be kept.

As Android uses Java as the application’s development language [1] and
Java is one of the most popular programming languages in the world [27], the IDE has to support Java as well. However, programmers often spent a substantial amount of time coding to cater for vulnerabilities exhibited in Java. One major obstacle to developer productivity is committing mistakes that the compiler can detect. In one case, the so-called NullPointerException has been called a 'billion dollar mistake' by its inventor Tony Hoarse, due to its frequent occurrences in numerous Java projects [15]. Although the compiler/interpreter can differentiate between nullable and non-nullable types, the decision to make all object types nullable delegates the task of type safety to the developer instead of the compiler. Compared with newer programming languages such as Python and Ruby, Java is verbose; it uses more lines of code to represent the same concept and often contains boilerplate code. Furthermore, Java is built on Object-Oriented Programming (OOP) concepts, leading to an unnatural integration for alternative programming paradigms. In response to Java's shortcomings, different alternative Java Virtual Machine (JVM) languages, such as Scala and Groovy, have been developed. While they avert the problems of Java, the need to compile down to JVM bytecode causes long compilation and linking time, thereby reducing developers productivity and desire to use these languages. Part of the goal of this project is to build a new programming language that addresses these shortcomings yet does not hinder developers work.

Therefore, a new programming language, Java+, is introduced into this cloud IDE platform. The new programming language not only aims to reduce verbosity and boilerplate code, but also encourages the use of different programming paradigms. It is also designed to prevent developers from making mistakes the compiler/interpreter can detect, maintaining a higher level of performance and efficiency. If even some developers are less interested in learning a new language, Java+ can still benefit these users with the “smart preview” function. In a typical IDE, developers can only toggle the visibility of code blocks (i.e. hide/unhide them), which allows them to focus attention on the important sections but does not allow them to refer back to the hidden blocks. In a smart preview, Java+ code logically equivalent to the underlying Java code will be displayed to the developer, giving them now have a bird’s-eye view of different components. Since Java+ code is often many times shorter than the equivalent Java code, this avoids the issue of cluttering the screen when the developer previews the code. We envision that the gentle introduction to Java+ will lead developers to adopt it over time.

Banerjee et al. [2] defines Everything as a Service (XaaS) as a type of cloud computing under which a single form of or, more often, a combination of services are provided online [23, 7, 14], and these services are easily accessible, independent of device and location, often with flexible scalability,
customizability and reusability [26]. By packaging many services into one platform, the IDE platform inherits the XaaS ethos, of which the services can be summarized into five major categories:

Platform as a Service (PaaS) defines a type of cloud platform where customers create and deploy their solutions using the resource pool supplied by the platform provider [20, 22]. In BuildApp, a Docker container has been allocated to each organisation for customers to create/develop their own projects, as well as compile and deploy them inside the container for testing.

Furthermore, BuildApp provides a Software As a Service (SaaS) for software project teams who require collaborative application development. SaaS is a type of service where customers can use without setting up their own self-host applications [24, 11, 20]. We intend that customers will be able purchase BuildApp services on a subscription basis. With a centrally-hosted architecture, BuildApp frees the customer from worrying about about security, storage and network problems.

BuildApp not only provides code collaboration, but collaboration on a broader basis with Communication as a Service (CaaS). The integrated project management service allows direct communication between developers and their clients. Developers can view feedback and issues side-by-side their working code, allowing their to make needed changes easier.

Finally, BuildApp provides Debugging as a Service (DaaS) for analysts and project managers. When mobile applications are compiled and run on testing devices, logs and screenshots/screen recordings can be stored in the platform. The developer can then use these data for analysis and visualizing to triage the issue and look for causes. The captured screenshots sent by the agent application can also be utilized by business analysts or developers to produce a higher quality of documentation. Overall, the detailed collection of debugging information permits a more measurable, quantifiable and traceable testing process.
Chapter 3

Existing Work

First, there are well known traditional IDEs such as Eclipse and Intellij, whose disadvantages as compared with cloud IDEs have been discussed in the Objectives and Benefits section.

Then, comparing current competing cloud-based IDEs in the market such as, Cloud9, Nitrious.io and Codeanywhere, they lack several key features of our IDE. The process of deployment is clunky and inefficient, with no innovation in this area as compared with traditional IDEs. Since developers spend considerable time building different versions of their apps and then deploying them to devices for testing, developers' productivity is not significantly increased. Furthermore, they do not address the problem of the limitations on developer productivity due to constraints of existing programming languages. New programming languages that aim at increasing productivity such as Scala and Groovy also exist, but they are usually not supported in these cloud IDEs. They also have other issues discouraging developers from using them, such as having slow compilation or interpretation time, or taking up significant amount of space in an application package. The IDE and new programming language integration tries to avoid the existing problems present in current solutions.
Chapter 4

Scope

In this project, three components that make up the whole software system will be developed. First, a web-based IDE with coding, compilation and debugging functionalities for Android and Java software development, which can support a large number of users, will be implemented. A runtime debugging and testing tool that can be installed into Android devices, allowing users to run their Java applications on our server and to preview the compiled application in their Android smartphones, will also be created. Last but not least, a new programming language and its interpreter or compiler will be developed.
Chapter 5

Design and Implementation

The overall process model this project will follow is the Agile procedure. In this procedure, components will be divided into tasks and assigned priorities. During every sprint, tasks with the highest priority will be assigned to each member for completion. At the end of each sprint, the team should be able to develop a usable product and will gather feedback. According to the feedback received, the team can re-assign priorities to existing and new tasks and start a new sprint. Adopting this methodology allows the team to move on quickly. As there are many components in this project, the following sections will describe the design and procedures for each components in detail.

5.1 Platform setup

The development server *fyp-pc06* has been assigned to our team by the Department of Computer Science. After setting up configurations of the server, virtual machines were created and tested.

5.2 Web-based IDE front-end

All key features of the front-end have been completed, such as the login screen page in figure 5.1 and the catalog screen page 5.2, where users can login and create/load projects respectively.

On the code page, the editor is fully collaborative, as seen in figure 5.3. Users can see what other users are editing, such as the red tag indicating that the other user is editing.

The editor is highly customisable with different themes and different services showing depending on the user’s actions. In figure 5.4, the user is using a dark theme and is testing his code. He can see the statuses of the testing
Since a direct replacement of one user’s edits with another user’s edits may result in lost edits, collaborative editing is enabled with operational transformation (OT), introduced by Ellis and Gibbs [10] and later refined by other researchers prevents this issue.
5.3 Web-based IDE back-end

5.3.1 Use of Go language as implementation language

After evaluating four different back-end programming languages, namely PHP, Go, Ruby and Python on maturity, nature of variable types, nature
of language, whether the language provides native concurrency support and whether the stack size is dynamic, Go was chosen as the implementation language. Table 5.1 shows the results of our findings.

<table>
<thead>
<tr>
<th>Language</th>
<th>PHP</th>
<th>Go</th>
<th>Ruby</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature of Variable Types</td>
<td>Dynamically typed</td>
<td>Statically typed</td>
<td>Dynamically typed</td>
<td>Dynamically typed</td>
</tr>
<tr>
<td>Nature of Language</td>
<td>Interpreted</td>
<td>Compiled</td>
<td>Interpreted</td>
<td>Interpreted</td>
</tr>
<tr>
<td>Native Concurrency Support</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic Stack Size</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of PHP, Go, Python and Ruby for the implementation of back-end processing language

5.3.2 Database

MongoDB has been chosen as the database system.

5.3.3 Compiling and executing environment

Docker[8] is used to provide an compilation and executing environment for users.

5.4 Agent application

Figure 5.5 shows the agent application login interface for which after users can register devices. It also can fetch applications from the server and install them, where it will prompt the user as in figure 5.6.

Moreover, the agent handles collections of debugging information such as screen recording and bug reporting as illustrated in figures 5.7 and 5.8.

5.5 Security design

Although only a single physical server is used to implement the BuildApp platform, any unused ports that can be pointed by the public IP are blocked by the middleware SSL proxy server before data reaches the server. Furthermore, each compilation job and compiled executable of an organization
is run in the pre-assigned Docker container which can only be accessed by the users of that organization. Therefore, the data of each organization is isolated from each other.

5.6 New programming language

Before development, XText was selected as the framework of implementation. Then, the design and coding of the new programming language was done in two parts, with the core component were created first to complete the Java+ to Java translation. Then, the language was integrated into the online IDE so that users could directly edit Java+ files and see the translation results in their browser.
5.6.1 Selection of XText as framework

One approach to creating a new programming language is to perform modifications to an existing programming language such as Python or Ruby, and then integrate this adopted language with Java. There are numerous advantages with this language design; the large amount of learning resources supporting these established languages renders a manageable learning curve for the majority of developers, thereby encouraging adaptation of this new language. Furthermore, since these languages already have a mature implementation, time needed to create a working adaptation is likely reduced compared to implementing a completely new programming language.

Despite these advantages, there are major shortcomings that prevent this approach from achieving the stated goals for the new programming language.
Since the syntax of Java and the modified programming language greatly differ, programmers will need to mentally switch contexts between the two languages as they alternatively edit source files encoded in Java and the new programming language, reducing their productivity.

A more serious issue is the penalty brought by runtime interoperability and translation. As shown in figure 5.9, Java classes exist in Java runtime (the green box) and classes of the new programming language exists in a different runtime (the yellow box). When data or functions flow from classes of one runtime to classes of another runtime, a procedure called translation happens, allowing the classes to communicate across the runtime boundaries. This procedure will need to run every time the boundaries are crossed. Furthermore, this procedure grows in size as more classes need to be translated into a different runtime. The cumulative time and space penalties in a medium to large application would bring a visible impact to the performance of the application and discourage developers from using the new programming language.

Therefore, another language design will be used, called the thin-layer approach as illustrated in figure 5.10. In this design, the new programming language is a thin layer on top of Java, meaning that the syntax diverges only slightly from Java. Differences in syntax are mainly for the purpose of avoiding well-known design pitfalls of Java. Due to the similarity, the translation from the new programming language back to Java or Java bytecode require less time and effort. This avoids the issues associated with integrating an adopted language.

Based on the decision to use a thin layer approach, there are two more technical aspects to consider before implementing the new programming language. Table 5.2 shows the advantages and disadvantages of the two types of output that can be generated from interpreting/compiling the new programming language, namely Java source code and Java bytecode, while table 5.3 evaluates the 5 different interpreter or compiler frameworks, namely LLVM,
Figure 5.10: The thin-layer design creates a new programming language which has a syntax that only slightly differs from that of Java.

PyPy, MPS, Graal/Truffle and XText, on key criteria such as maturity, scale of Java support and Java interoperability support.

Comparing the advantages and disadvantages of outputting Java source code versus outputting Java bytecode, both have advantages that can contribute to the stated goals of the new programming language. The plug-and-play nature of Java source code output means that developers do not need to create special configurations for the new programming language and do not need to worry that incorporation of the new code into existing applications will create additional errors. Overall, this allows the developer to be more efficient. On the other hand, Java bytecode is more performant in nature as a second round of compilation is not needed. Balancing the benefits and disadvantages of the two approaches, outputting Java is a better solution for the project for two main reasons. Outputting Java allows a plug-and-play approach where the output can simply be dropped into wherever the old code existed. This can ensure that future maintainers unable to read Java+ can still maintain the project. Since LLVM, PyPy¹ and Graal/Truffle output Java bytecode [25, 21], they are removed from consideration and only MPS and XText are considered.

Although MPS and XText are similar in the criteria applied, the approaches taken by MPS and XText in creating the language are different. XText uses a more traditional approach that uses a parser to parse textual

¹Early PyPy versions had a Java bytecode backend but it was removed due to lack of interest
<table>
<thead>
<tr>
<th>Output</th>
<th>Java source code</th>
<th>Java bytecode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>- Plug &amp; play</td>
<td>- Performant</td>
</tr>
<tr>
<td></td>
<td>- Can utilize existing Java libraries and runtime</td>
<td>- Easier to create than Java output in some cases</td>
</tr>
<tr>
<td></td>
<td>- Can integrate with any environment that runs Java with further configuration</td>
<td></td>
</tr>
<tr>
<td>Disadvantages</td>
<td>- Encourages manual editing of output</td>
<td>- Cannot integrate with existing environment without configuration</td>
</tr>
<tr>
<td></td>
<td>- Requires more time to create</td>
<td>- Reduces portability of output</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison of Java source code output and Java bytecode output from the new programming language

files. On the other hand, MPS uses a projection-based editor [28] that appears like text but actually allows editing of the underlying AST structure. While the additional layer of abstraction in MPS provides certain advantages, such as easier embedding of languages in existing languages, this does not provide much benefit since a new language is created. However, the underlying format is specific to MPS and is harder to reuse outside of the workbench. XText, on the other hand, allows the language to be reused outside the workbench as other tools can manipulate the textual files. Therefore, XText will be the language workbench used to develop the language.

5.6.2 Core components

After deciding to proceed with XText, the syntax of the new programming language was developed. Listing 5.1 shows a sample of the syntax.

Listing 5.1: Sample of new programming language syntax

```java
package app {
    entity MainActivity extends Activity {
        def onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.main)
        }

        def onCreateOptionsMenu(Menu menu) {
            super.onCreateOptionsMenu(menu)
        }

        def setContentView(R.layout.main) {
        }
    }
}
```
A key concern is that the new language should be able to represent most concepts in Java, in order for it to be a suitable replacement language for Java. Therefore, it heavily derives from Java in order to maintain high compatibility with Java. However, inspiration is also taken from Javascript which many developers are familiar with and is generally considered to be less verbose than Java.

After designing the syntax, a grammar defining the language was created. The grammar is in part defined by new rules and tokens, and in part delegated to XBase, a useful mini-language that provides many building blocks. Since the grammar has to cover many use cases as a general purpose language, delegated allows the grammar and development of the other components in the processing pipeline to be kept at a manageable state.

When the grammar is defined, XText can use the grammar to derive a parser that converts the input code into an abstract syntax tree (AST). Then, a translator program, which is developed by the programmer, converts

<table>
<thead>
<tr>
<th>Framework</th>
<th>LLVM</th>
<th>PyPy</th>
<th>MPS</th>
<th>Graal/Truffle</th>
<th>XText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td>Bytecode</td>
<td>Bytecode</td>
<td>Source code</td>
<td>Bytecode</td>
<td>Source code</td>
</tr>
<tr>
<td>Maturity</td>
<td>Stable</td>
<td>Stable</td>
<td>Stable</td>
<td>Under R&amp;D</td>
<td>Stable</td>
</tr>
<tr>
<td>Scale of Java back-end support</td>
<td>Partial</td>
<td>Full</td>
<td>Full</td>
<td>Full</td>
<td>Full</td>
</tr>
<tr>
<td>Just in time optimization in JVM</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>Java interoperability</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.3: Comparison of LLVM, PyPy, MPS, Graal/Truffle and XText for the implementation of a new programming language

```java
// Inflate the menu
getMenuInflater().inflate(R.menu.main, menu)
val button = findViewById(R.id.button1)
button.setOnClickListener(
    view ->
        Toast.makeText(MainActivity.this, "Button Clicked",
        Toast.LENGTH_SHORT).show())

return true

}
}

```
the AST into the AST of Java. With this component, XText can take the outputted Java AST and turn it back into the ultimate Java output using pretty-printing. The full pipeline is shown in figure 5.11.

![Diagram showing the translation pipeline from Java+ to Java](image)

**Figure 5.11: Illustration of translation pipeline from Java+ to Java**

Since the parser is automatically generated from the user-defined grammar and the pretty-printer is handled by the framework, the other component in this pipeline that has been developed is the translator. In the framework, the translator walks through the Java+ AST, and at each node delegates the translation to the function that converts the Java+ node into a Java node. Therefore for each new node that is defined, a conversion function to Java node(s) is needed.

Therefore, with the two components developed, the full pipeline has been completed where the user can type code in Java+ and get Java output. Listings 5.2 and 5.3 show a sample piece of Java+ code and its outputted Java code respectively.

**Listing 5.2: Java+ code sample**

```java
import java.util.ArrayList
import java.util.Arrays
import java.util.Collections
package app {
    entity Person extends Comparable<Person> {
        var age : int
        val friends = new ArrayList<Person>()
        def youngestFriendByDistance(int distance): Person {
            Collections.min(
```
if (distance == 0) friends else
friends.map([Person p |
    Collections.max(Arrays.asList(p, p.
    youngestFriendByDistance
daistance=1))))
}

def compareTo(Person p) {
    this.age - p.age
}

Listing 5.3: Java output for Java+ code in listing 5.2

package app;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import org.eclipse.xtext.xbase.lib.Functions.Function1;
import org.eclipse.xtext.xbase.lib.ListExtensions;

@SuppressWarnings("all")
public class Person implements Comparable<Person> {
    private int age;

    public int getAge() {
        return this.age;
    }

    public void setAge(final int age) {
        this.age = age;
    }

    private final ArrayList<Person> friends = new ArrayList<Person>();

    public ArrayList<Person> getFriends() {
        return this.friends;
    }
}
public void setFriends(final ArrayList<Person> friends) {
    this.friends = friends;
}

public Person youngestFriendByDistance(final int distance) {
    List<Person> _xifexpression = null;
    if ((distance == 0)) {
        _xifexpression = this.friends;
    } else {
        final Function1<Person, Person> _function = new Function1<
            Person, Person>() {
            @Override
            public Person apply(final Person p) {
                Person _youngestFriendByDistance = p;
                _youngestFriendByDistance((distance - 1));
                List<Person> _asList = Arrays.<Person>asList(p,
                    _youngestFriendByDistance);
                return Collections.<Person>max(_asList);
            }
        }
        _xifexpression = ListExtensions.<Person, Person>map(this.
            friends, _function);
    }
    return Collections.<Person>min(_xifexpression);
}

public int compareTo(final Person p) {
    return (this.age - p.age);
}

As listing 5.3 shows, aside from some necessary Xbase polyfills (additional code which provides functionalities that are not in the native Java environment), the outputted code is plain Java and is formatted in a way as if it was written by a human. Readable Java output permits programmers to directly edit the Java code if the need arises, whether if the original Java+ code is lost or the programmer does not want to learn Java+.

5.6.3 Integration with web IDE

After the full translation pipeline was created, integration with the web IDE began. As the framework provided web integration that exposed an editor
integrated with the pipeline, we first tried to integrate the Java+ editor with the existing editor inside the web IDE. However, it was discovered that the Javascript API interfaces of the two editors were too different for integration. Therefore, it was decided that web IDE would step in for the Java+ editor and make API calls to the Java+ editor backend. Since the calls were not well documented, reverse-engineering by triggering different actions in the Java+ editor was needed to discover the format of the calls. Furthermore, as the web IDE needed to load all Java source files generated from the Java+ source file, which not provided by the default backend, a custom handler was installed on the server. Using the validation and generation handler in the backend, the frontend can coordinate the transformation from a Java+ document to Java source file(s). A complete diagram of the integration workflow is shown in figure 5.12.

Figure 5.12: Illustration of the integration workflow from Java+ to Java on the web IDE
Chapter 6

Deliverables

Deliverables are divided into 4 major components, namely the IDE front-end, the IDE back-end, the helper Android application and the new programming language.

6.1 Web-based IDE front-end

The web-based IDE front-end should include the following features:

- Coding editor with syntax and application program interface (API) hinting and searching
- Drag-and-drop graphical user interface (GUI) editor
- Seamless collaborative coding
- Git version control integration
- Code and application templates
- Remote delivery of executables to user devices
- Displaying execution result of Java terminal-based programs
- User account management for teams

6.2 Web-based IDE back-end

The back-end infrastructure will provide a separated environment for each user. It will also compile code, store files as well as host the database system. The back-end infrastructure will also include a communication API for
the front-end infrastructure to obtain user data and files as well as execute commands in the separated environment of each user. Besides, the back-end server will also communicate with the helper Android application, which will be explained in the next section.

6.3 Agent application

The agent application helps developers to deploy Android applications to mobile devices for testing and debugging, by both developers and clients. Developers first sign in BuildApp Agent with their account in the IDE. For clients, signing in using an invitation code from developers. After signing in, the device will be bound to the account of the user or bound to a project. To deploy the application, the developers can select a device from a list of devices they have added to their account or bound to the project. The application package file will be pushed to the selected device and a prompt will appear for the device users to confirm the installation. After the installation is completed, the device user can start and test that application. BuildApp Agent would also collect debugging messages from the device and send them back to the server in order to display the messages to developers for debugging purposes. Moreover, BuildApp Agent also provides functions includes screen capturing and recording. During testing, the screen of the device could be recorded and send to developers. In this way, whenever there is a bug, developers can investigate into the videos and fix the issues accordingly.

6.4 New programming language

The new programming language will consist of three to four parts: the specification, the Java to Java+ translator, the Java+ to Java translator and integration with the cloud IDE, including the smart preview function.
Chapter 7

Division of Labour

Wong Man Chun Kelvin and Shum Chi Chung Charlie were responsible for the development of the cloud IDE. Back-end infrastructure development was shared between them. Kelvin was in charge of the creation of the server logic and database structure. Front-end development was Kelvins responsibility including designing the UI and creating the coding editor, syntax hinting tool, API hinting and searching tool, drag-and-drop GUI editor, collaborative coding tool, version control integration and code/application templates. Charlie was in charge of all other tasks including setting up the Android build system [5] and Java compiler on the Docker containers [8], creating the connection between the IDE and the build systems, implementing the Agent App on Android to receive Android application packages with debugging tools and feedback collection features, and logic on the IDE website to receive and display messages and feedback sent from the Android devices.

Poon Stirling Yeu and Wong Man Chun Kelvin took responsibility for the development of the new programming language. Stirling was in charge of feature design and implementation, support for alternative paradigms and the implementation of compilation back-end targeting Java/JVM bytecode/assembly code/Javascript or interpreter. Kelvin assisted in the integration of the Java+ service with the web IDE.
Chapter 8

Results and Challenges

In the ICT awards, we were honoured to receive the Best Student Innovation (Tertiary or above) Certificate of Merit award in the Hong Kong ICT Awards 2016. During the presentation, we received quite positive feedback from the judges where they were impressed with the revolutionary new features, such as the collaborative editing and push to device features, and felt that it would bring great improvements to productivity for developers. However, some felt that collaborative editing would be confusing for programmers since other developers could change other parts of the code without one noticing, and one judge asked, “if two people edited the same line at the same time, how would you resolve the edits being mixed up?” After thinking about this issue for some time, we believe we have a solution as discussed in the next section.

Despite the fact that Java+ is a prototypical language, some advantages over Java have been shown. To test my implementation of Java+, I took many snippets of Java code from the web to rewrite into Java+ and test the resulting Java output. It was observed that the Java+ code was often 20-50% more compact, in terms of lines of code, as compared with the Java code. When I showed Java+ code to other developers, many commented that it seemed to be a “cleaner” Java and looked somewhat like Javascript. However, when asked whether they would want to use Java+ over Java, some said they would still prefer Java as Java is the “tried and tested” language, or that they did not have the time or opportunity to learn a new programming language; however, they added that they might be most inclined to use it if they had more time to study it or see it more matured and developed. To overcome this inertia, some suggestions are devised in the next section.

For the implementation of the new programming language, several challenges were encountered. Since each new language needs to implement several key components correctly for the language to work, the API interface of XText is rather large. Therefore, I spent significant time learning this new
API and how I could use it. Furthermore, the web integration support of XText is quite new and not very well documented. Hence, I underestimated the time needed to integrate it into the main IDE when I discovered that the two editors were not compatible on the Javascript level, and I had to reverse-engineer the API calls of the Java+ editor.
Chapter 9

Future plans

After Java+ was successfully developed and shown to potential users, we received feedback that while Java+ appeared to be less verbose than Java and could increase productivity, users may not feel the need to learn Java+ as it would take some time to learn the new language. Therefore, to ease the learning curve, the new programming language could serve a dual purpose of providing a “smart preview” function.

9.1 Smart preview

In a typical IDE, programmers often are changing code within a small section of the source code, meaning that the code surrounding the edited code may not be relevant to the programmer. Therefore, they can collapse the unneeded sections to focus on the section being edited. However, a full collapse of the unneeded sections means that if the programmer wants to refer back to them, he/she needs to extend the entire section. Since logically equivalent Java+ code is less verbose than its corresponding Java code, showing the Java+ code can help the programmer refer to less important sections without cluttering the editor space.

Unfortunately, this new purpose had not be planned when the decision to use XText was made. Since the implementation of Java+ using the XText framework had been made, I spent some time investigating the API documents to investigate whether a translation component from Java to Java+ could be built. However, it was discovered since the scope of the grammar of Java is much larger than Java+, i.e. more types of Java AST nodes than Java+ AST nodes, the translator from Java to Java+ would require many more translation functions. With the time limitations, implementing this translator would be infeasible. To implement “smart preview”, I would need
to port the current grammar and Java+ to Java translator to a more suitable framework before implementing the Java to Java+ translator.

9.2 Collaborative editing algorithm

For general collaborative editing of documents, OT is usually sufficient. However, as reflected by the judges in ICT Awards presentation, this may lead to inefficiencies or even errors in a coding environment. For example, if Alice only needs to change one line of code while Bob needs to change 100 lines of code, even after Alice finish making her change, Bob is still making his changes, making the entire piece of code possibly not compilable until Bob has finished making all his changes. This means that Alice needs to wait for some time before she can test her one changed line. Goldman, Little, and Miller [12] propose “collaborative editing with semantics” to resolve this issue. Unlike OT where everyone can see the result of everyone’s edits immediately, all users see a “error-free” version of everyone’s edits. After a certain number of pending edits have been collected, the largest subset of these edits that can be cleanly applied to last-known error-free document \( E \) is merged into \( E \) to form \( E' \) and propagated to all users, and at the same time replaces the old \( E \) as the last-known error-free document. Note that \( E' \) is not the final version reflected to users since each user see the results of his/her pending edits applied to \( E' \). An brief illustration of this algorithm is illustrated in figure 9.1.

Figure 9.1: Illustration of the collaborative editing with semantics algorithm

However, only having compilable/error-free as an indicator of correctness
may not be sufficient for all purposes. For example, let us assume that Alice is working on class A and Bob is working on class B. Within class A, function X should only return an integer from 0 to 100 since it represents a score in an exam. Working on that assumption, Bob changes some code in B to calculate an average score. However, when all changes have been propagated, Bob discovers that his code is incorrect. It turns out that some exams have a maximum score of 120 causing the algorithm in B to malfunction.

While Bob can easily discover the issue in this example, logical errors are more difficult to catch than compilation errors. Having other forms of version control cannot entirely prevent this issue; they only make it easier to catch, such as when Bob discovers incorrect answers when he merges Alice’s branch and his branch. Therefore, the way to detect logical errors automatically is for the developer to write tests to ensure the logical constraints are obeyed by the code.

Within the collaborative editing with semantics algorithm, I propose that if the developer can provide tests, when the largest subset is calculated, not only does the resulting document \( E' \) have to be error-free, but also has to pass the tests successfully. Depending on the rigidity required by the programmers, a hard-fail or soft-fail approach can be used, as shown in figure 9.2.

In the hard-fail approach, passing the tests is a compulsory criteria for selecting the largest subset. While this ensures that the error-free document is always free of logical errors, passing the tests is a higher barrier than being compilable, especially if the tests are too stringent or even wrong. Edits that are constantly logically wrong will not be merged and users may think that the real-time collaboration is not working. For the soft-fail approach, passing the tests is an optional criteria for selecting the largest subset, but users will be informed that the selection is not the best and the tests failed by \( E' \). While \( E' \) may not be error-free, users have the ability to determine for themselves whether the code actually contains logical errors or the tests are wrong.

While we do not have sufficient time in this iteration to finish developing this enhanced algorithm, for the next iteration, we would like to incorporate this algorithm into OT to make the editor more user-friendly.

Alternatively, some users may still be used to version control and may be hesitant to switch to collaborative editing completely. At the same time, we want beginners and teams to be able to just jump in and start coding. Therefore, we plan to introduce a toggle between version control and collaborative editing for advanced users to switch between the two at their will.
Figure 9.2: Illustration of the collaborative editing with semantics and logically testing algorithm. Note that while \{ea1, ea2, eb2, ec3\} is the largest subset that is compilable, it fails the user-provided tests so that next largest compilable and logically correct subset is \{ea1, ea2, eb2\}.

35
Chapter 10

Conclusion

In this paper, the need for a machine-independent cloud-based IDE and a new programming language has been identified. The IDE and new programming language aim to reduce the barrier of entry to using the IDE and increase developer productivity.

The proposed technology stack and implementation of the IDE and new programming language have been described in this report. While challenges have been encountered when creating these key components and issues raised in some of the our approaches, the new technologies incorporated in this IDE have allowed us to investigate how different algorithms and solutions can yield new approaches for better collaboration amongst developers. We look forward to implementing our future plans in the next iteration.
List of Figures

5.1 Screenshot of login screen page .................................................. 14
5.2 Screenshot of catalog screen page .................................................. 14
5.3 Screenshot of code screen page with another user editing ............... 15
5.4 Screenshot of code screen page with dark theme and debugging panels .......................................................... 15
5.5 Login page of agent application .................................................... 17
5.6 The agent application requests permission to install a deployed application “Currency App” onto the testing device ............. 17
5.7 The agent application provides a floating menu with options, including to start the screen recording or to send a bug report .... 18
5.8 The agent application notifies that the screen is being recorded ......... 18
5.9 Translation happens when classes transfer data across the run-time boundaries ......................................................... 19
5.10 The thin-layer design creates a new programming language which has a syntax that only slightly differs from that of Java 20
5.11 Illustration of translation pipeline from Java+ to Java ...................... 23
5.12 Illustration of the integration workflow from Java+ to Java on the web IDE .............................................................. 26

9.1 Illustration of the collaborative editing with semantics algorithm 33
9.2 Illustration of the collaborative editing with semantics and logically testing algorithm. Note that while \{ea1, ea2, eb2, ec3\} is the largest subset that is compilable, it fails the user-provided tests so that next largest compilable and logically correct subset is \{ea1, ea2, eb2\} .................................................... 35

37
List of Tables

5.1 Comparison of PHP, Go, Python and Ruby for the implementation of back-end processing language . . . . . . . . . . . . . . . . . . . . . . . . . 16
5.2 Comparison of Java source code output and Java bytecode output from the new programming language . . . . . . . . . . . . . . . . . . . . . . . 21
5.3 Comparison of LLVM, PyPy, MPS, Graal/Truffle and XText for the implementation of a new programming language . . . . . . 22
Listings

5.1 Sample of new programming language syntax . . . . . . . . . 21
5.2 Java+ code sample . . . . . . . . . . . . . . . . . . . . . . . 23
5.3 Java output for Java+ code in listing 5.2 . . . . . . . . . . . 24
Acronyms

API  application program interface. 27
AST  abstract syntax tree. 22
CaaS  Communication as a Service. 10
DaaS  Debugging as a Service. 10
GUI  graphical user interface. 27
IDE  Integrated Development Environment. 4, 6, 8–13, 15, 25–29, 36
JVM  Java Virtual Machine. 9
OOP  Object-Oriented Programming. 9
OT  operational transformation. 14, 33
PaaS  Platform as a Service. 10
SaaS  Software As a Service. 10
XaaS  Everything as a Service. 9, 10
Bibliography


