Flutter for GarageGarner

Flutter provides an implementation of Dart that is particularly conducive to mobile applications. As compared to Python, Java, and OCaml, it presents distinct and significant advantages in many areas. The downsides mostly come from the fact that the language is still in alpha, and thus framework and support may not be fully developed. This analysis will perform a language-by-language analysis, highlighting in order the advantages of Dart over Java, OCaml, and Python, respectively. It will end by discussing the disadvantages of using Flutter for this application.

Java

Since this project will require development of an application for androids, the first language that often comes to mind is Java. While applications can be run by a virtual machine on the mobile device, it is slightly different than standard Java, and so might require some changes as opposed to the OS version of this application. This is a problem which Dart addresses, providing one syntax and semantics that requires as little as 5% code change to be run on both iOS and android devices.[1] It is hard to overstate the advantages of having entire teams working on the same framework in terms of both code sharing and co-development. As an added bonus, the syntax of Dart is incredibly clean and well organized and can be structured in a very sequential way. With Java, UI code can require XML to properly separate the building of an interface from the program code, but Dart's syntax requires no such extension, simplifying greatly.[2] This is especially important when we consider than a user interface on an android will often want to use Lisp code, which may require some very ugly FFI written in Java to integrate into the program, and which can be avoided completely by the usage of Dart.

OCaml

Having seen the complications of attempting to utilize varying Java environments, we may decide to consider the usage of a strongly typed, statically compiled language like OCaml. This should immediately bring to mind, however, the fact that compilation of OCaml programs can be rough. The strict typing can mean that programs that are logically sound may be rejected, and this can increase the time per iteration of developing a program greatly. Additionally, Dart allows for static compilation anyway. It has type inference and annotation, giving us the error catching capabilities of OCaml, in combination with the ability to run in an interpreter, which allows for short iteration times and fast integration of changes.[3]

OCaml faces another significant issue in terms of android development; function calls are slow. To allow for currying, OCaml must push the frame of a returning function onto the heap, an action which increases the latency of embedded function calls. This is particularly problematic on an android – the environment of an interface can change drastically and quickly, whether it be by rotation of the phone or suspension of an app, and android's support for this involves many function calls. The device destroys all elements of the interface and reconstructs them when necessary. Invoking these getters and setters could be especially difficult if OCaml is used to

[1] why-we-chose-flutter-and-how-its-changed-our-company-for-the-better

[2] why-native-app-developers-should-take-a-serious-look-at-flutter-

[3] why-flutter-uses-dart

implement our program, as this could greatly increase the time spent for even small changes and cause lots of jank when attempting to run in real time.

Python

In attempting to migrate computation from the server side to the client side, Python is a natural choice. It is the language with the strongest support for TensorFlow and provides a very simple interface for asynchronous function calls, allowing us to perform necessary computations in tandem with any other operations necessary. Unfortunately, Python has a difficult time running on mobile devices. When running asynchronous code, it has been known to chew up massive amounts of memory. Memory is in short supply on a phone, and with the asynchronous calls involved in server communication, awaits may be necessary, depending of course on the structure of the program. This would place a limitation on the number of concurrent Tasks we could run. While this is conceivably reconcilable, Dart proves a much better option in this respect.

Dart utilizes a mixed memory management approach that incorporates many of the best aspects of many approaches. For one, it uses a nursery divided into two subsections, one of which is active at a time, allowing for unused space to be reclaimed extremely quickly. This is a modified version of the approach used by Java. It also incorporates the two-stage approach used by modern Python; the second stage of Dart garbage collection involves a mark and sweep, which is run seldomly, since the structure of our mobile app will mean that the program utilizes many short-lived objects which may exist completely within the nursery. Additionally, Dart processes run as Isolates, so named because their memory is isolated from that of other isolates. These provide the distinct advantage of being able to schedule garbage collection extremely efficiently without any worry of locking or race conditions.[4]

Disadvantages

From the presentation of this report so far, it may seem that Dart provides exclusively advantages above all other language choices. Dart does, however, present some disadvantages, although these are admittedly non-extreme; since Dart incorporates attributes of many different languages, it tends to be familiar and easy to utilize for programmers any programmers familiar with the fundamentals of object-oriented programming. It provides a lot of syntactic sugar that allows programs to be clearly expressed, especially when documented with comments. This brings us to one of the major disadvantages of Dart, however; while the documentation of Dart is extremely thorough and clear where it can be found, the language is still in alpha, and thus does not contain all the libraries one may require. Thankfully for our project, there is support for tflite, but we may stumble upon missing libraries which require us to write boiler-plate code that may not be required had we written the program in another language. This will improve over time of course, as Google continues to develop the language. The other of our inconveniences is similar; since Dart and Flutter are relatively obscure at the moment, the framework necessary for development tends to be relatively large. While this is rather inconvenient, it is yet

[4] flutter-dont-fear-the-garbage-collector

again not a huge deal. Additionally and similarly, we may see incompatibility or changes in syntax which may render our current understanding irrelevant as the language further develops.

Conclusion

Dart provides the flexibility of a dynamically run and interpreted language with the robustness and speed of a statically typed and compiled language. It combines the best aspects of many modern object-oriented paradigms to form a language that supports all aspects of app development. This is not surprising, having come from Google. Dart has an extremely efficient memory allocation and garbage collection scheme that supports quick allocation of many temporary objects – perfect for a widget-based mobile app environment. Dart can also be compiled to ARM machine code which can be run as top speed on mobile devices. A pre-trained neural network could provide what I consider to be top quality support for our application.

References

A tour of the Dart language [Internet]. Dart. [cited 2020Jun5]. Available from: https://dart.dev/guides/language/language-tour

Jain A. Why native app developers should take a serious look at Flutter [Internet]. Hacker Noon. 2020 [cited 2020Jun5]. Available from: https://hackernoon.com/why-native-app-developers-should-take-a-serious-look-at-flutter-e97361a1c073

Leler W. Why Flutter Uses Dart [Internet]. Hacker Noon. 2020 [cited 2020Jun5]. Available from: https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf

Smith M. Why we chose Flutter and how it's changed our company for the better. [Internet]. Medium. Medium; 2018 [cited 2020Jun5]. Available from: https://medium.com/@matthew.smith_66715/why-we-chose-flutter-and-how-its-changed-our-company-for-the-better-271ddd25da60

Sullivan M. Flutter: Don't Fear the Garbage Collector [Internet]. Medium. Flutter; 2019 [cited 2020Jun5]. Available from: https://medium.com/flutter/flutter-dont-fear-the-garbage-collector-d69b3ff1ca30