Dart for Mobile Development

CS 131 Homework 6

Stephanie Doan

I. Abstract

With the rise of smart phones that have undoubtedly impacted every facet of our daily lives, efficient mobile app development has garnered much attention from the developer community. Debates between Android and iOS, native and React Native, and client-side and server-side computation have been brought up to determine the best approach for mobile development. One of the newest and increasingly popular technologies in this domain is Flutter, an open-source user interface software development kit. It is written in Dart, a programming language created for making mobile, server, desktop, and web applications with focus on elegant and easily produced UI. Here we investigate both Dart and Flutter and their unique features and fit for general mobile development, especially the specific app under investigation: GarageGarner, a mobile garage sale assistant that involves computer vision and machine learning to help shoppers a more informed shopping experience.

II. Background

A. Dart and Flutter

Introduced by Google in 2011, Dart is a client-optimized language that focuses on smooth development of user interfaces for apps on various platforms. Particularly, Dart can compile into native machine code for mobile, server, or desktop. Dart Native has a Dart VM with the option for just in time (JIT) compilation, which offers pure interpretation as will as optimization at runtime, or ahead of time (AOT) compilation into x64 or ARM machine code. The AOT compiler is used for production so that the compiled native code starts and runs the app seamlessly and efficiently, as machine code interfaces directly with the hardware. Speed is crucial to a mobile applications user experience. In addition, Dart is a great choice for web applications as it compiles into JavaScript code which runs in the browser. Dart Web offers dartdevc, a development-time compiler that is optimized for the fastest developer turnaound, and dart2js, a production-time compiler into fast and deployable JavaScript through methods such as elimination of dead code which wastes memory and computing power.

Dart also offers a garbage collector in its runtime to allocate and deallocate memory, which is necessary and useful because of the short average lifetime of objects for this use case. For mobile applications, Stateless Widgets rendered, destroyed, and remade as the app the user interacts with the application interface very rapidly and changes the app's state in short time intervals. This is especially advantageous for complex apps with many small widgets that are created and destroyed often. Dart's garbage collector is designed to maintain high application performance, by scheduling the full operation of the collector when the Flutter app user is inactive for a certain period of time. Detection of and response to user activity is crucial to Dart's efficiency and should alleviate programmers of worrying about the efficiency of running a fairly active garbage collector in the mobile application.

Interestingly, Dart is single-threaded, carrying out operations sequentially, and each operation is guaranteed to not be interrupted by another. As seen before in NodeJS and Python's Asyncio library, Dart has an event loop that handles the execution of asynchronous functions. Therefore, it is important to avoid taxing the main thread with operations that tend to cause bottlenecks, like large network requests or file IO or heavy computation. However, threading can be achieved through the concept of an Isolate, which is essentially a thread itself. They are given their own memory to avoid data races and ensure thread safety, unlike operations in the main thread that share memory important for communication between operations. Isolates instead communicate by passing messages to each other.

Flutter is the UI toolkit built on top of Dart, offering an efficient mobile development experience with all the advantages of the language. Native

interfaces can be built with customizable and quickly rendering widgets, built with aesthetically pleasing built-in Cupertino and Material Design components, seamless scrolling and motion that are often overlooked in their role to create a seamless user experience.

B. Language

The Dart language offers interesting nuances that are worth examining. Expanding upon the previously mentioned single-threaded system, Dart uses specifically the Future and Stream classes for asynchronous programming. The Future object is used to represent the result of a delayed computation. When declared, a value or error is not yet available but shall be in the future, which is received to trigger a callback that handles that value when it materializes. This is analogous to the Promise in JavaScript, which is applied to similar use cases of web, server, and mobile development. The Stream object, on the other hand, is a source for asynchronous events, which can be classified as either data or error events, followed by a done event that signifies the end of the stream. StreamSubscription objects subscribe to such Streams, which can be single-subscription streams, which allow only one listener often for high amounts of continuous data, or broadcast streams made for multiple independent listeners for independent events.

Dart is characteristically an object oriented and class-based language. An object is an instantiation of a class, the "blueprint", which is defined by its variables and methods. The syntax for defining a class is as follows:

```
class Pencil {
      var color;

      describe() {
            print("This pencil is $
{color}");
      }
}
```

An object of the Pencil class can be instantiated as such:

```
var myPencil = new Pencil("blue");
```

The "new" keyword is followed by the class name, and in parenthesis is the argument to the constructor, assigning the value "blue" to the "color" variable. Then, variables and methods are accessed via a dot between the object name and the desired variable or method name.

```
//variable
myPencil.color
//method
myPencil.describe()
```

There are not specific access modifiers that mark methods and variables as private or public as in C++ or Java, but an underscore that precedes declaration signifies that that method or variable is private to the library.

In addition, the Dart type system uses sound typing, which means both runtime and static type checking are used to check that a variable's value corresponds to its static type. Therefore, the programmer is guaranteed that their checked program will never unexpectedly be in an invalid state because Dart's static analyzer is able to find type-related bugs at compile time. This makes the code more readable, maintainable, and efficient for AOT compilation.

III. Comparison with Languages

A. Python

Compared to Python, Dart is a newer language that has been used less frequently, so the developer support is more sparse as a result. Python is also a dynamically typed and interpreted language, which offers flexibility for the cost of speed, while Dart is statically typed as previously mentioned. Dart's asynchronous functions is analogous to Python's Asyncio library, both running queued operations in an event loop.

B. Java

Java and Dart similarly use sound typing to perform checking a compile time, which yields many benefits for performance. Although Dart is single-threaded, Java offers extensive abilities and options in multithreading whereas Dart's Isolates strictly allocate independent memory for each thread and inter thread

communication in the form of messages. As previously mentioned, Java offers different access modifies such as public, private and protected, but Dart does not offer that level of control.

C. OCaml

The most noticeable difference between OCaml and Dart is that OCaml's classification as a functional programming language while Dart is object oriented. OCaml emphasizes recursion and efficiency for intensive computation, but Dart is tailored to quickly create and destroy rapidly changing UI components. In terms of typing, Dart differs from OCaml because of Dart's dynamic type objects, the "var" declaration keyword indicating that the variable will be the initializer type when it is initialized, and its value is able to change until the "final" keyword which indicates that the variable's value will no longer change during its lifetime.

IV. Analysis

The main decision to be made about GarageGarner is whether to perform heavy machine learning computation in the server or on the mobile device itself using TensorFlow Lite. The first option poses the disadvantage of high network bandwidth, but the performance of the mobile application itself would not be greatly affected because of Dart's power in asynchronous programming that prevents these large network requests from becoming a bandwidth. The second option, however, poses a danger that the mobile application consumes too much memory and computation during runtime. This is remedied by Dart's AOT compilation that takes place before deployment, saving space and app startup time when it is being used.

Dart is fairly well equipped to deal with the downsides of either of these two options. In addition to these specific language features, Dart and Flutter offer a clean developer and elegant user experience, and GarageGarner will benefit from the transition to Flutter.

Write an executive summary that compares and contrasts Dart to the other languages, and gives Dart's strengths and weaknesses, along with problems for this application.

V. Conclusion

Dart and Flutter prove to be very promising for app development, as it is highly focused on a positive user experience through elegant design decisions and fundamental speed optimizations. The attention to detail on the user side does not sacrifice the developer experience, through features like JIT compilation and a development-time compiler for the web. The cross-platform app development is extremely valuable, as web and mobile products often go hand in hand for businesses, and they would ideally share many parts of the development experience. A comparable technology is the React and React Native frameworks created by Facebook, which offers richer developer support but pales in terms of developer productivity, built-in features as opposed to third-party libraries, and testing capabilities. Some predict Flutter to be the future of mobile app development, its advantages attributed to the favorable designs of Dart.

VI. Works Cited

[1] Parth, Dave. "Single Thread Dart, What?" *Medium*, Mobile Studio India, 1 June 2020.
Sullivan, Matt. "Flutter: Don't Fear the Garbage Collector." *Medium*, Flutter, 4 Jan. 2019.

[2] "Asynchronous Programming: Streams." *Dart*, dart.dev/tutorials/language/streams.

[3] "Python vs Dart Detailed Comparison as of 2020." *Slant*, www.slant.co/versus/110/383/.

[4] "Flutter vs React Native: A Developer's Perspective." *Nevercode*, nevercode.io/blog/flutter-vs-react-native-a-developers-perspective/.

[5]"Dart Object: W3Schools: Tutorialspoint." *W3Adda*, 31 May 2019, www.w3adda.com/dart-tutorial/