CS 131 Homework 6: Evaluation of Dart as a Tool for GarageGarner

Computer Science 131, Programming Languages, University of California Los Angeles

Abstract

When building an application as complex as *GarageGarner*— a tool that utilizes machine learning and computer vision to find the best deals at garage sales for users— not only is *how* the app built is important, but also *what* the app is built with. *Flutter*, a development kit released by Google in 2017, combined with the client-optimized programming language *Dart*, have been gaining traction in the application development world. Dart's ease of use, flexibility, and reliability, while unfortunately compromising some performance relative to other native mobile languages, makes it a strong option for developing GarageGarner.

1. Introduction

GarageGarner will be an application to find people the best deals at garage sales, and it will on their mobile devices. The technology will exploit machine learning, and specifically computer vision, by examining price tags and products and returning the best deals it finds. TensorFlow Lite has been selected to run the machine learning model, but a language needs to be chosen to run the applications interface and execute the model in. Dart, a language that runs in the Flutter interface toolkit is the proposed language [1].

1.1 GarageGarner

A consumer will be able to walk into any garage sale, open the GarageGarner application on their smartphone, take a panoramic photo of the products and price signs, and be given back the best deals there within a reasonable time. All of the computation will be done locally, on each person's own smartphone, to eliminate the hassle of network latency, and to allow users to use the app offline.

The application will run faster than in it did in the old design that involved uploading the images to a central server, processing the machine learning algorithms on the server, and then returning the results. Both server bottlenecks and network latency delays will be avoided by running the application locally [1].

1.2 Flutter and Dart

The application is proposed to be implemented using the Flutter UI software development kit created by Google, and programmed with Dart, a client-optimized language for applications also developed by Google [1]. Dart is marked by its ease of use: as a language that descended from the ALGOL language family, Dart has C-style syntax [2]. It smartly handles memory using a generational garbage collection and allocation scheme [3]. In addition, Dart is flexible and general. Dart is object oriented, and supports abstract classes, interfaces, interfaced objects, nested functions and more. All variables are objects, as well as functions and even null. Dart can be type checked statically or dynamically, or both— and can be compiled pre-execution or just-in-time [4]. Dart's unique combination of convenient features put it ahead of most languages available today.

2. Features of Dart

Dart, being a high level, modern programming language, is marked by its extreme ease of use, flexibility, generality and reliability, but relatively slow performance.

2.1 Ease of Use

Dart is one of the many descendants of the ALGOL language family, alongside C, Java, C#, JavaScript and others. Its syntax is extremely similar to C and therefore will be very easy for programmers to pick up in a matter of days.

Additionally, Dart is loaded with features that boost convenience. Dart allows *method cascading style [7]*:

```
a..b()
..c();
```

Is the same as:

```
a.b()
a.c();
```

Dart also features *optional parameters [7]*. For example, the following function implementation is allowed:

```
void printThings(String requiredArgument, [String optionalArgument]){
        print('$requiredArgument     $optionalArgument');
}
```

The two ways the one function can be called:

```
printThings('required');
printThings('required', 'optional');
```

Optional parameters decrease the size of source code by combining methods that would need to be overloaded in most languages, and decrease the time spent writing programs.

Dart features lots of convenient syntactic sugar for commonly written patterns in code, such as:

```
class Point {
        num x,y;
        Point(this.x, this.y); //Constructor
}
```

The constructor is not merely a declaration, but is also an implementation that sets the member variables x and y [7].

Besides convenient syntax, Dart also features lots of core features built into the language that ease its use.

Dart can be *dynamically typed*, meaning the type of any variable does not have to be explicitly declared when writing programs. Hyper-managing types when programming can be tedious, and not having to worry about it can speed up writing programs [6].

Memory in Dart is *garbage-collected*, meaning objects declared dynamically do not have to be manually deleted by a programmer [3].

Dart can be *just-in-time* compiled, meaning a full compilation of the program for each execution is not required. Instead, the lines or section of code that is about to run are compiled. JIT compilation allows for fast prototyping and testing of code currently being developed [4].

Dart also features *hot reload*, a feature that helps you quickly and easily experiment, build UIs, add features, and fix bugs. Hot reloading usually takes only a second, and it injects modified source code while preserving the state of your program [8].

Dart also *powerful*, *modern libraries* that come with the language by default.

For example, Dart has an *async library* very similar to Python's asyncio, but it comes by default, so every machine running Dart has the capability to execute programs asynchronously. The async library is also extremely easy to use, similarly to Python it uses `await` and `async` keywords [5].

As another example, Dart has the type *iterable* that comes standard with the language. Iterables can be easily mutated:

```
print([1,2,3].where((x) => x.isOdd).map((x) => x+10));
> (11,13)
```

The same functionality in Java is possible, but would require downloading FluentIterable, because it is not part of the SDK [4].

Lastly, Dart has the *Dynamic* type keyword, which allows a data type to be inferred at run time. The dynamic keyword is implicitly given to variables that are not manually statically typed, but can put in templated places as well. The following allows a list to have multiple types in it:

```
List<dynamic> canAddAnyType;
```

2.2 Flexibility

Dart was designed with flexibility in mind and it shows

Dart is *object oriented* and *class based*. Everything in Dart is an object, including numbers, null, and even functions. Dart supports inheritance, abstract classes, templated objects, and everything that marks a modern OOP language [4].

It also supports *reified generic objects*, meaning generic types carry their type arguments at run time, unlike in languages like Java. So the following is possible [7]:

```
new List<String>() is List<int>     // false
new List<String>() is List          // true
new List<String>() is List<dynamic> // same as line above
new List() is List<dynamic>         // true, these are exactly the same
```

Dart can be either dynamically typed or statically typed, or both, depending if the programmer declares a type before the variable (which is completely optional always).

Dart's list of convenient features that allows abstraction and adaptability continues, including interfaces, nested functions, etc.

2.3 Generality

Dart's rules are applied generally. As previously stated, all values are objects, there are no primitive types to worry about. When researching this language, no exceptions to any rules were shown. Exceptions to rules are not too common in well written and frequently used languages, so this does not necessarily make Dart stand out far too much.

2.4 Performance

Dart conveniently offers two runtime modes:

Checked Mode is a developer-friendly mode that performs just-in-time compilation, switching source code to machine code on the fly to reduce compilation time. It is slower, and performs type checking on variables declared with types [9].

Production Mode is the default mode of a Dart program. Optimized for speed, production mode compiles to machine code and ignores assert statements and static types [9].

However, even in production mode, Dart is much slower than lower level languages like Java. According to "The computer Language Benchmarks Game" Dart typically performs about 3 times slower at a task than a Java program, with significantly higher memory usage [10]. See: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/dart-java.html for complete results. Dart's subpar performance when compared to Java could be a deal breaker for this CPU intensive project. However, this was the benchmark available online, and other sources claim Dart is extremely fast, so the evaluation of Dart's speed should be tested by the team before development.

2.4 Reliability

Dart is *type safe*. While types are mandatory, type annotation are optional because of Dart's type inference. Dart's type system is *sound*, it ensures that at runtime a value can never evaluate to one that does not match the expression's static type [10].

Dart also uses *automatic garbage collection*, meaning it automatically manages the heap and will delete inaccessible dynamically declared data, and clear the heap at the end of execution. An automatic garbage collector increase reliability by preventing memory leaks [3].

Dart, similar to Python, is single threaded. While this makes concurrent programming impossible, it increases reliability by giving the developer more control over the execution. Because the exact ordering of execution is always certain, critical functions like animations executed to competition, can be ensured [4].

2. Strengths Specific For this Project

Besides Dart's ease of use, flexibility, and support for fast development cycles, its GUI focused nature and garbage collector lend itself to helping GarageGarner specifically.

2.1 Client Optimized and GUI Focused

Dart 2.0 was rebooted in 2018 with a focus on optimizing client-side development for web and mobile. Dart works extremely well on the two critical host operating systems for this application, iOS and Android— and works just as well on the web as a bonus, leaving the option to one day port a version onto the browser. The program also supports a *reactive* style of programming that is logical for mobile applications.

Additionally, Dart's UI markup is integrated into its code syntax— or in other words, programmers do not have to switch back and forth between a separate UI markup language and Dart when designing the app [12].

2.2 Generational Garbage Collector

For mobile applications, widgets are created as they are rendered on screen, and destroyed when they go off screen. Thousands of short-lived objects would constantly be created and destroyed for this project, and Dart's generational garbage collector is optimized for handling short term memory [3].

2.3. Flutter Tensorflow Lite

Dart's support for Flutter Tensorflow Lite is an extremely convenient way to design and execute machine learning models on devices. One AI engineer who had no app development was able to get an image classifier that utilizes deep learning working within a week. See:
https://medium.com/innovation-incubator/real-time-image-classification-on-android-using-flutter-tflite-2674f03caf0f
for his story [13].

3. Weaknesses Specific For this Project

3.1 Speed

Dart's main weakness (if any), is its speed. As outlined in section 2.4, Dart is benchmarked at significantly slower than Java, another language used to develop mobile applications. Because that was the benchmark that was available, if the team moves forward developing GarageGarner in Dart, it would be smart to prototype ML models and benchmark the performance against other languages before creating the application.

3.1 Familiarity

While Dart is an extremely easy language to pick up, if the team is efficient at developing mobile applications in other languages, they might not be as productive for the first couple weeks as they familiarize themselves with a new one. Languages like Swift, Java, and Kotlin are far more frequently used to develop apps, and it is more likely the team is already familiar with these languages than with Dart and the Flutter environment [14].

3. Dart vs Other Languages

3.1 Ocaml

OCaml is at its core a functional programming language, setting itself apart from the get-go to Dart, an imperative language. OCaml compiles to bytecode while Dart fully compiles when instructed. OCaml is statically type-checked while Dart is not, unless written in that fashion. Both languages have automatic garbage collectors. OCaml would not be a good choice for this project because its functional programming style does not support complex, abstract data structures. Dart's support of a reactive programming style and its ability to create GUI's makes it the obvious winner compared to OCaml [15].

3.2 Java

Both Java and Dart are imperative, object-oriented, have automatic garbage collectors, and are used to build apps. However, Dart is likely to have faster development cycles with its type inference and prototyping compilation mode. But Java will run much faster most likely with its easy multi-threading capabilities, and primitive types that put itself closer to the hardware. With GarageGarner, the main concern is speed, so sacrificing Darts attractive modern qualities that give it an ease of use for Java (an already easy to use language) is most likely worth it if it the benchmarks in 2.4 correctly reflect the performance difference in GarageGarner [16].

3.3 Python

Python and Dart at a first glance are extremely similar languages. They are both known for their "modern" programming style qualities: ease of use, type inference, powerful standard libraries, automatic garbage collection. They both are object oriented. While Python has TensorFlow as well, Python is only interpreted, meaning with almost all certainty it will be far too slow to execute machine learning models on mobile devices. It also is extremely infrequently used to create applications— creating a smooth user experience would be very difficult using Python.

4. Conclusion

Dart's ease of use, fast development cycles, and suitability for GUI's make it a strong option for GarageGarner. However, its measured weak performance is a serious concern, and benchmark testing would need to be performed with prototype ML models before it would be the choice over Java for GarageGarner.

5. Sources

[1] http://web.cs.ucla.edu/classes/winter20/cs131/hw/hw6.html
[2] https://en.wikipedia.org/wiki/Flutter_(software)
[3] https://medium.com/flutter/flutter-dont-fear-the-garbage-collector-d69b3ff1ca30
[4] https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf
[5] https://dart.dev/codelabs/async-await
[6] https://dart.dev/guides/language/sound-dart
[7] https://dart.dev/guides/language/language-tour
[8] https://flutter.dev/docs/development/tools/hot-reload
[9] https://subscription.packtpub.com/book/web_development/9781783989621/1/ch01lvl1sec10/setting-up-the-checked-and-production-modes
[10] https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/dart-java.html
[10] https://dart.dev/guides/language/sound-dart
[11] https://dart.dev/guides/language/sound-dart
[12] https://medium.com/dartlang/announcing-dart-2-80ba01f43b6
[13] https://medium.com/innovation-incubator/real-time-image-classification-on-android-using-flutter-tflite-2674f03caf0f
[14] https://www.androidauthority.com/develop-android-apps-languages-learn-391008/
[15] https://en.wikipedia.org/wiki/OCaml
[16] https://en.wikipedia.org/wiki/Java