CAB202 Asynchronous Programming with Async Await

A survey on its usage in c# and javascript and the paradigms related

Anhad Ahuja

June 12, 2022

1 Introduction

Async and await is fundamentally syntax to support modern asynchronous programming paradigms. Imperative or procedural programming is highly sought after (paradigm wise) in the modern software industry. Imperative programming offers a more obvious flow of state and progression to the program and is thus known as algorithmic programnming.

This api does not seem to be specifically aimed at performance for reasons explored in the implementation section however, it does seem ver applicaple to developers that wanted to do the same thing consistently which is to wait for actions that are out of thier control. Black box approaches being supported by this feature further aids to the facade paradigm heavily induced within the c# object oriented ecosystem.

2 How async works under the hodo in c#

2.1 State Machine implementation

2.2 SynchronizationContext

The context that a piece of code gets run on is

```
      public HomeViewModel()
        {
            SelectedIndex = 0;
            Games = new ObservableCollection<Game>(Db.games);
            DispatcherTimer timer = new DispatcherTimer();
        }
```

3 History

3.1 How it was introduced

Interestingly the concept of async await which fundamentally attempts to remove declaritive paradigms, actually originates from $F\#$ all the way back in 2007.[?] Monads are a functional programming paradigm aimed to assist in reducing the visual plumbing required to get from one data type to another. In f# computational expressions were used to unwrap the monad to the consumable datatype inside of the expression. This can be seen in the following example:

```
open Microsoft.FSharp.Control.COmmonExtensions

let fetchURLAsync url =
  async {
    let req = WebRequest.Create(Uri(url))
    use! resp = req.AsyncGetResponse()
    use
  }
```

This came in the form of monads to support a context where

3.2 Previous paradigms

3.2.1 APM

This implementation can be awknowledged as using the message passing paradigm

3.2.2 EAP

4 Using lazy evaluation of enumerables: `IAsyncEnumarable`

5 UI based example and how to avoid system locking

Locks up Ui, conventionally heavy computationaly or IO based operation that locks up UI thread is undesirable. It could be possible to create a new thread using system API and delegate tasks to that, but that has a lot of computational and programmer overhead. Generally not a trivial matter especially to sync up content and data.

6 `Task.Run` based example

7 Additional quirks and useful features of Task based await

7.1 Cancellation tokens

7.2 `IProgress`

8 Comparison to javascript

9 Javascript similiar example using `Promise`

References

[1] S. Don, "Introducing f asynchronous workflows," *Microsoft Developer Network*, 2007.