State Machine을 사용한 Flutter 어플리케이션 Bloc 상태관리 모델링

Modeling Flutter Application State Management Using State Machine

요 약

Flutter 어플리케이션 상태관리를 모델링 하는데 State Machine을 적용함으로써 Flutter 어플리케이션 상태 관리 방식을 높은 단계의 추상화로 이해할 수 있으며 UI에서 발생하는 모든 events를 파악할 수 있다. 또한 Flutter 어플리케이션의 Cross Platform 특성에 맞추어 달라지는 UI events를 파악하고 어플리케이션에 사용되는 Widget을 올바른 상태로 모델링 할 수 있다.

1. 서 론

Flutter[1]는 Google에서 배포한 오픈소스 크로스 플랫폼 GUI 어플리케이션 프레임워크이다. 특징으로 하나의 단일한 소스코드로 모바일, 웹, 데스크톱 환경에서 모두 사용가능한 GUI를 작성할 수 있어 멀티플랫폼 서비스를 개발하고자 하는 개발자들에게 큰 관심을 받고 있다.[2] Flutter로 작성되는 GUI는 Widget[3]이라는 기본 단위로 구성 되어있다. Widget은 한번 빌드 되면 다시 화면을 빌드하지 않는 Stateless Widget과 사용자 혹은 어플리케이션의 이벤트를 통해 상태정보가 수정되는 Stateful Widget으로 나뉜다. 이 중 Stateful Widget은 사용자와 상호작용하는 대화형 어플리케이션 개발에 있어서 필수적으로 사용된다. Stateful Widget의 State 정보를 관리하고 체계화하는 것을 Flutter의 State Management라고한다.

State Management 개발을 편하게 하기 위하여 GetX, Bloc, MobX와 같이 다양한 State Management 패키지가 개발되어 사용 중이나, Flutter로 개발되는 어플리케이션의 Size와 Complexity가 증가하며 UI에 올바른 State를 배치하는 것은 더욱 어려운 일이 되고 있다.[4] 또한 Flutter는 Cross Platform을 지원하여 UI에서 발생하는 events가 달라지기 때문에 작동되는 플랫폼 별로 state 변화를 가져오는 events를 다르게 매핑해주어야 한다. 예를 들어 화면의 데이터를 새로 갱신하는 events가 웹에서는 새로고침 버튼이지만 모바일에서는 아래로 끌어당기는 스와이프로 event가 변경될 수 있다. Flutter State Management는 이러한 Cross Platform 특성을 고려하여 event와 UI의 visible 속성에 주의하여 설계되어야 한다.

본 논문에서는 State Machine을 사용하여 Flutter Application의 Bloc State Management[5]를 모델링하는 방법을 제시하였다. State Machine을 통해 Bloc State Management를 높은 추상 단계에서 살펴볼 수 있으며, 어플리케이션에서 발생하는 모든 이벤트를 확인할 수 있다. 또한 Flutter의 Cross Platform 특성을 반영하여 Bloc State Management의 Bloc이 플랫폼 별로 다른 이벤트를 수신하고, state 변경을 다르게 적용하는 것을 확인 가능하게 하였다.

2. 배경지식

Bloc State Management

Bloc State Management는 Flutter에서 사용가능한 State Management 패키지로서 Bloc, Cubit을 사용하여 UI의 이벤트를 수신하고 UI의 State를 변경한다. Bloc 패키지의 아키텍처는 그림1과 같다.

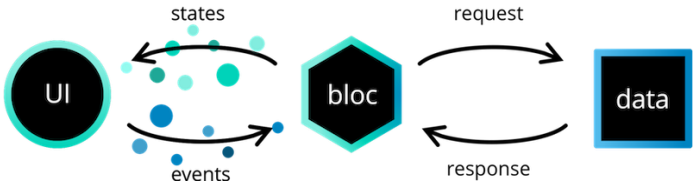

그림1. Bloc State Management Architecture

UI에서 events가 트리거 되면 Bloc은 data layer에 요청을 보내어 state의 새로운 값을 할당하고 이를 UI에 다시 전달한다. Bloc 패키지의 가장 큰 특징은 특정

state를 하나의 Bloc에서 전적으로 관리하기 때문에 해당 state를 사용하는 모든 UI가 매번 새로 상태관리 모듈을 작성할 필요 없이 해당 Bloc을 재사용 할 수 있다. State에 대한 Bloc의 책임과 역할이 분명하기 때문에 어플리케이션을 테스트하기에 다른 상태관리 보다 용이하다.

State Machine

State Machine[6]은 UML(Unified Modeling Language)로 표현되는 finite automata의 확장된 표현이다. 기존의 FSM(Finite State Machine)의 주요 한계점을 극복하기 위하여 제안되었다. 기존의 FSM이 특유의 형식 주의로 인하여 State Explosion 문제를 자주 겪는 방면에 UML State Machine은 Hierarchically nested states와 Orthogonal regions 등의 개념을 도입하여 중첩되거나 동시성이 요구되는 상태를 표현하는데 더욱 편리하다.

3. 관련연구

크로스 플랫폼 어플리케이션 혹은 모바일 어플리케이션의 Size와 Complexity가 증가함에 따라 객체 지향적인 개발방법에서 모델 지향적인 개발 방법을 적용하려는 여러 연구가 진행되었다. Mehreen Khan[7]은 모바일 어플리케이션 State Management를 위한 UML 프로필을 제안하였으며, Ayoub Sabraoui[8]은 DSL(Domain Specific Language)를 사용하여 모바일 어플리케이션을 개발하기 위한 개발방법론을 제안하였다. 기존의 연구는 모바일 어플리케이션에 중점을 두어 접근하였으나, Flutter는 모바일과 웹, 데스크톱 환경을 모두 지원하는 GUI 프레임워크로 이를 고려하여 모델 주도 개발 방법을 접근할 필요가 있다.

4. Bloc State Management를 위한 State Machine 모델링 방법 제안

Bloc State Management를 State Machine으로 표현하기 위해서 그림 2와 같은 모델링 방법을 제안한다.

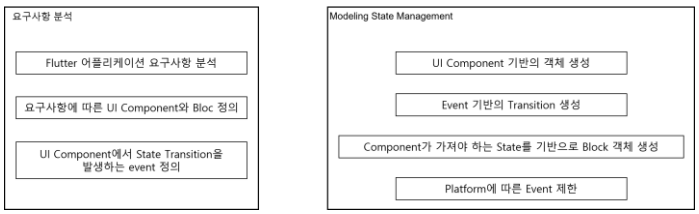

그림2 Bloc State Management 모델링 과정

모델링 방법은 크게 2가지 단계로 나뉜다. 요구사항 분석 단계에서는 Flutter 어플리케이션의 요구사항을 분석하여 Flutter 어플리케이션에 사용되는 UI Component와 event를 식별한다. 이 과정에서 식별된 UI Component는 State Machine의 객체로 사용되며 사용자가 UI와 상호작용으로 발생하는 event는 State Transition을 정의하는데 사용된다. Cross Platform의 특성을 반영하여 같은 기능으로 작동하는 UI Component일지라도 각 Platform에서 실제 작동가능한 event를 식별해주어야 한다. 그림 3은 Floating Button을 누르면 화면상의 숫자를 1씩 증가해주는 Increment 어플리케이션의 요구사항을 분석한 예시이다.

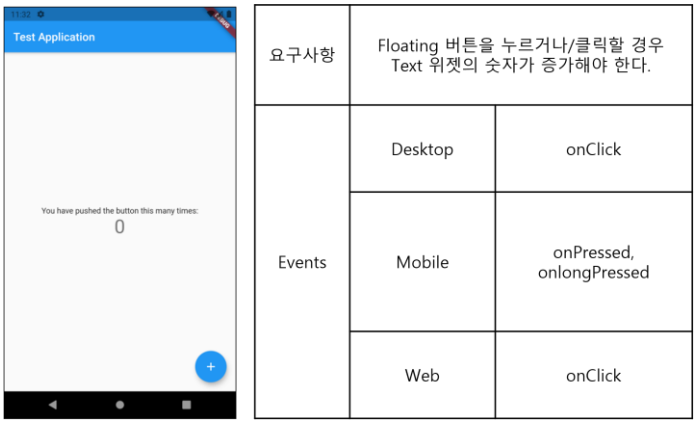

그림3. Increment App 예시

모델링 단계에서 요구사항으로부터 식별된 UI컴포넌트를 State Chart의 객체로 사용하며 events를 기반으로 Transition으로 사용하며, State 별로 Block 객체를 생성한다. 단 이때 Platform 별로 events를 별도로 구분하기 위하여 플랫폼 별로 Guard가 존재해야 한다. 제안된 방법을 통해 개발자는 Platform 별로 어떤 event가 Bloc에 수신되는지, 해당 Bloc을 사용하고 있는 Widget은 어떤 Widget인지 시각적으로 파악할 수 있으며, 단순히 코드로 작성된 State Management에 비하여 Widget과 Bloc간의 연결관계를 분명하게 파악할 수 있다. 그림 4는 Increment App의 Bloc State Management를 모델링 방법에 따라 State Machine으로 표현한 것이다.

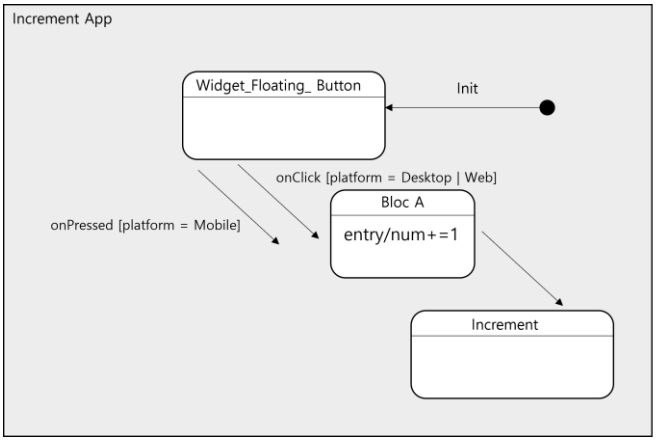

그림 4. Increment app Bloc State Management State Chart

4. 결론 및 향후 연구

본 논문에서는 Flutter 어플리케이션의 Bloc State Management 모델링에 State Chart를 활용함으로써 Flutter 어플리케이션이 서로 다른 플랫폼에서 실행될 경우 해당 Bloc을 Trigger하는 event의 종류를 구분하고 Bloc이 Trigger되는 event와 Bloc이 State를 변경시키는 위젯의 관계를 분명하게 하여 Bloc State Management 모델을 작성하였다. 그러나 이번 논문에서 제안한 모델링의 한계점은 State의 수와 상호작용하는 Widget이 적은 Increment app을 사용하여 실제 Cross Platform 어플리케이션에서 State의 수와 Widget의 수가 증가할 경우 State Chart로 표현 가능할지 알 수 없으며 좀더 규모와 복잡도가 큰 어플리케이션을 대상으로 한 Case Study가 진행되어야 한다.

차후 연구 목표는 각 플랫폼 간 guard 되어야하는 event를 구분하고 동기/비동기 상황에서 State Machine을 사용한 State Management 모델링이 유효한지 확인해볼 예정이다. 또한 이번에 제안된 방법을 통해 생성된 모델이 유효한 모델인지 검증할 것이다.

참고 자료

[1]Flutter Github 저장소, https://github.com/flutter/flutter 2022,06,23 확인

[2] Growth of Flutter, https://medium.flutterdevs.com/the-growth-of-flutter-development-3years-after-the-birth-of-alpha-78baee809dff , 2022,06,23

[3] Introduce to Widget, https://docs.flutter.dev/development/ui/widgets-intro, 2022,06,23

[4]

[5] Architecture of Bloc, https://bloclibrary.dev/#/architecture, 2022,06,23

[6] OMG (February 2009). "OMG Unified Modeling Language (OMG UML), Superstructure Version 2.2"

[7] A Model Driven Approach for State Management in Mobile Applications, Mehreen Khan, Farooque Azam,Muhammad Waseem Anwar,Fatima Samea, Mudassar Adeel Ahmed, 2019 8th International Conference on Software and Computer Applications, February 2019 Pages 315-319

[8] MDD Approach for Mobile Applications Based On DSL, Ayoub Sabraoui, Anas Abouzahra, Karim Afdel Mustapha Machkour, 2019 International Conference of Computer Sc 22-24 July 2019ience and Renewable Energies (ICCSRE),