Open Issues and/or Actions

No open issues.

Table of Contents

[1 DOCUMENT INTRODUCTION 3](#_Toc483325054)

[1.1 Purpose 3](#_Toc483325055)

[1.2 Scope 3](#_Toc483325056)

[1.3 References 3](#_Toc483325058)

[1.4 Terminology & Abbreviations 3](#_Toc483325059)

[2 Overview 4](#_Toc483325060)

[3 Architecture 4](#_Toc483325081)

[3.1 Architectural Overview 4](#_Toc483325082)

[3.1.1 Concepts 4](#_Toc483325083)

[3.2 Architectural Priniciples 6](#_Toc483325084)

[3.2.1 Dependency Management 7](#_Toc483325085)

[3.2.2 Choice of SDK v/s Gateway 8](#_Toc483325086)

[3.2.3 Architectural Patterns 9](#_Toc483325087)

[3.2.4 uApp Development Rules 10](#_Toc483325088)

[3.2.5 Execution Architecture 10](#_Toc483325090)

[3.2.6 Rules and Guidelines 14](#_Toc483325095)

[3.3 Architecture Views 14](#_Toc483325096)

[3.3.1 System decomposition view 14](#_Toc483325097)

[3.3.2 Software View 17](#_Toc483325102)

[3.3.3 Process view 17](#_Toc483325107)

[4 Allocation of Quality Aspects 17](#_Toc483325111)

[5 Design Details 18](#_Toc483325112)

[5.1 Navigation Logic 18](#_Toc483325113)

[5.2 Configurable UI. 18](#_Toc483325114)

[5.3 External Interfaces 19](#_Toc483325117)

[5.4 Internal Interfaces 19](#_Toc483325125)

[6 Revision History 21](#_Toc483325132)

[7 Approval 21](#_Toc483325133)

# DOCUMENT INTRODUCTION

## Purpose

The purpose of this document is to identify the architectural blocks and explain the top level design of Reference App for iOS and Android. The document provides an overview of the various functional blocks of reference app, design choices made and guidelines to few elements. This forms a reference for detailed design and implementation.

## Scope

This document describes the high level software architecture of the Reference app. The contents of this document apply to both the iOS and Android.

The interface and detailed design for each component is out of scope of this document.

Below are the core components of Reference app, it is built for Android phones with version 4.4 and above and IOS 8.4 and above.

* Data
* Navigation and Logic
* Content
* Common Components

## References

| **Reference** | **Identification** | **Title / additional remarks** |
| --- | --- | --- |
| SRS | REF000001 | Software requirement specification |
|  |  |  |
|  |  |  |

## Terminology & Abbreviations

| **Terminology & Abbreviations** | **Description/Definition** |
| --- | --- |
| App | Mobile application |
| AppInfra | Mobile application infrastructure library |
| uAppFWK | Micro app application framework |
| Coco | Common component |
| OS | Operating System |
| Platform | The platform is an integrated collection of components that can be used to support multiple solutions across multiple business groups. |
| Proposition | Synonym of solution, with the connotation that the focus is a more on the value as in “value proposition” and less on the product. |

# Overview

Reference app is a sample app for propositions to start with from which new apps can be derived quickly by adding or stripping component.

It contains example code to demonstrate usage of various common components.

This document describes the high level software architecture of the Reference app. The contents of this document is applicable to both iOS and Android.

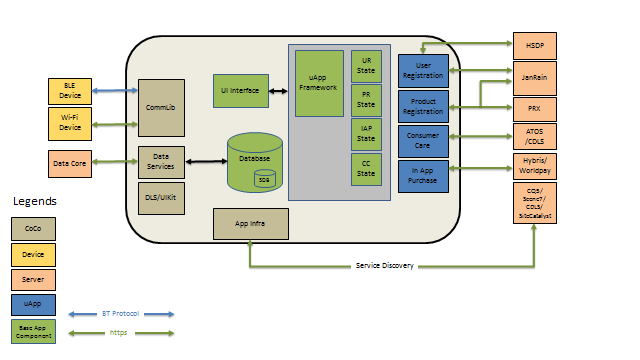
The interface and detailed design for each component is out of scope of this document.



# Architecture

## Architectural Overview

Ref App is a sample application which forms as base platform for all propositions to develop and maintain features both generic and specific to the propositions.



### Concepts

The Ref App architecture consists of five elements, namely:

* Ref App components
* uApps
* Servers
* Devices

#### Ref App Components:

Ref App components are the elements that reside within the inner source archive.

It includes:

* UI Interface
* Database
* uApp States

User interface in ref app shows some sample screens in a user journey like splash, settings, onboarding welcome screens etc. This can be taken by propositions and modify as per their requirement.

Database currently which is part of ref app is part of data services implementation to communicate to data core. Propositions are expected to create their own implementation.

uApp states represents typically an abstraction on top of micro app stanadard interface which is used to transition from one state to another using flow manager.

Onboarding screens, settings and splash are also considered as states to help in smoother navigation using Flow manager even though it is not a micro app.

#### CoCos:

Common Components (CoCos) are reusable components with specific functionality. Some of the CoCos are:

* CommLib
* Data Services
* DLS/UIKit
* AppInfra

#### uApps:

uApps are common components with its own user interface and functionality. Internally, the uApps communicate with their respective servers to store and/or fetch data as per requirement. Some of the uApps are:

* User Registration
* Product Registration
* Consumer Care
* In App Purchase

uApps are integrated in ref app through standard interfaces and represented as states which connects through flow manager to form the user journey.

#### Servers:

Each component communicates with the servers to store and fetch data for various purposes. For example:

* Storing user details
* Storing product details
* Maintaining transaction history
* Maintaining user moments and consent
* Storing/Reading consumer care data
* Data processing

The servers in the ref app and their purpose are given below.

| **Server** | **Purpose** |
| --- | --- |
| HSDP | Health suit digital platform which provides various services including identity management etc |
| JanRain | Storing user account details and product details |
| Hybris | Storing shopping details, such as:   * Product list * Pricing information * Cart details * Shipping and billing address details * Partial card information   It also redirects the user to World pay for payments and instructs the backend to ship the product if user has purchased it. |
| PRX | Storing product information |
| Autonomy | Saving latitude and longitude information of Philips service centers/retailers. |
| CQ5 | Content Management. |
| Scene7 | Storing media, such as images and videos. |
| CDLS | Storing Customer Care information like phone number, email link, chat link etc. |
| SiteCatalyst | Data analytics |
| DataCore | Storing user moments and consent |

#### Devices:

The CoCos interact with external devices, such as BlueTooth and WiFi devices to exchange data over a specific range.

Comm lib provides an abstraction to communicate to both BT and wifi devices using BT and DICOMM protocol respectively.

## Architectural Priniciples

Architectural Decisions are design decisions that address architecturally significant requirements. Architectural Decisions influence and impact the non-functional characteristics of a software. Each Architectural Decision describes a concrete, architecturally significant design issue for which several potential solutions exist. An Architectural Decision captures the result of a conscious, often collaborative option selection process and provides design rationale for the decision making outcome. Architectural Decisions concern a software system as a whole, or one or more of the core components of such a system.

There are many aspects of Architectural Decisions. While designing the architecture of Mobility Platform, the Architectural Decision aspects followed are:

* Dependency Management
* Choice of SDK v/s Gateway
* Architectural Patterns
* UApp Guidelines
* Execution Architecture

### Dependency Management

Dependency is to a broad term that refers to when software components depend on one another. Dependency in Mobility Platform is divided into two categories, namely External Libraries and Internal Libraries dependencies. Further, Internal Libraries dependency is divided into two categories, namely Mandatory Libraries and Other Internal Libraries.

#### Internal Libraries

All components and micro apps is delivered in the form of library which is then integrated as part of ref app.

Mobility Platform uses a number of internal libraries, such as CommLib, Consumer Care, AppInfra, UIKit, AppFramework, and so on. The version management of these libraries is done in collaboration with component teams and CI pipeline team.

Ref App allows interdependencies of these libraries. But circular dependency is not allowed between internal libraries.

Among the internal libraries, uApp framework, AppInfra and UIKit are mandatory libraries. All propositions must use these libraries.

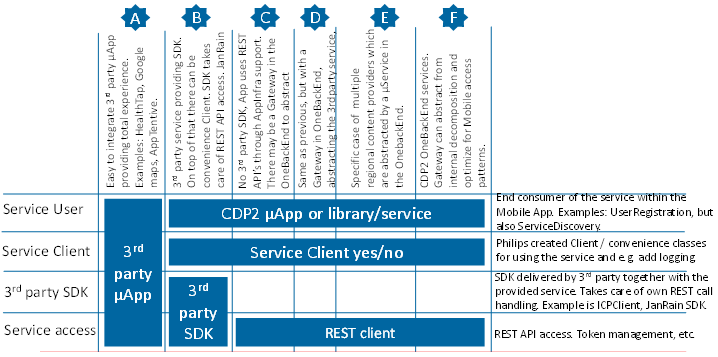
#### External Libraries

Mobility Platform uses a number of external libraries like Janrain, AF networking, Volley etc. Component teams needs to seek IP&S for approval before using any external third party libraries.

The choice of external and internal libraries needs a conscious approach. There are a few aspects that affect the choice of these libraries. Some of them are described below.

|  |  |
| --- | --- |
| External Library | Internal Library |
| Readily available. | Needs to be developed exclusively. |
| Cheaper in cost. | Costly, since cost of development is high. |
| No support available. | Support readily available. |
| Ecosystem updates may not be available. | Ecosystem updates always available. |

### Choice of SDK v/s Gateway



| **Cloud service** | **Owner** | **Comments** | **Model (🡪 direction)** | **µApp / Web URL** | **SDK / Client** | **REST API** |
| --- | --- | --- | --- | --- | --- | --- |
| CQ5 | PIL |  | C (+AppInfra Client) |  | By AppInfra | √ |
| Scene7 | PIL | use Scene7 for movies, not YouTube: native localization | A (player/viewer, other 3rd party than Scene7)) |  | URL directly to the player or viewer for images | √ |
| YouTube (PC) | 3rd party |  | A |  | YouTube SDK |  |
| PRX | PIL | PRX is created by IT | C (+AppInfra Client) |  | By AppInfra (PRXClient) | √ |
| Hybris | PIL |  | C (+in CoCo Client) |  | (Yes, but not used) | √ |
| WorldPay | PIL | Only Web URL to avoid need for PCI DSS compliance in CDP2 platform | A | Web URL | -- | √ (not used) |
| SiteCatalyst | PIL | Adobe clickstream analysis | B (+AppInfra Client) |  | Adobe SDK + AppInfra | √ (not used) |
| Test& Target | PIL | Adobe, used for A/B testing | B (+AppInfra Client) |  | Adobe SDK + AppInfra | √ (not used) |
| JanRain | PIL | Social logins are abstracted by JanRain. SDK forked by us. | B (+in CoCo Client) |  | Used. | √ |
| AppTentive | PIL | InApp feedback. | A&B (Client TBD) | µApp | Used to set states |  |
| HSDP user services | HSDP | (HSDP user services is partial abstraction for JanRain) | C 🡪 Client & Gateway (D) TBD |  | Special class in UR component | √ |
| DeviceCloud | HSDP |  | B (ICPClient + ‘wrapper’) |  | ICPClient | -- |
| DataCore | CDP2 |  | C 🡪 Client & Gateway (F) TBD |  | DataSynch + TBD | √ |
| CDLS | PIL | contact data lookup service from IT: name, e-mail, phone | -- |  | See PRX | PRX Gateway |
| BazaarVoice | PIL | not yet used. write/read reviews for Philips website | B |  | SDK |  |
| Autonomy | PIL | Find GPS of Philips near to you | C |  | -- | √ |
| FaceBook CC  FaceBook UR | 3rd party |  | A | URL to website | Not used | Not used |
| SalesForce | PIL | URL obtained through PRX | A | Only e-mail. | Not used | Not used |
| Social sharing | 3rd party |  | B |  | iOS/Android platform |  |

### Architectural Patterns

 Architectural patterns solve and delineate some essential cohesive elements of a software architecture. The architectural patterns being followed in ref app are:

* State Pattern
* MVP Pattern

#### State Pattern

The state pattern implements a state machine in an object-oriented way. With the state pattern, a state machine is implemented by implementing each individual state as a derived class of the state pattern interface, and implementing state transitions by invoking methods defined by the pattern's superclass.

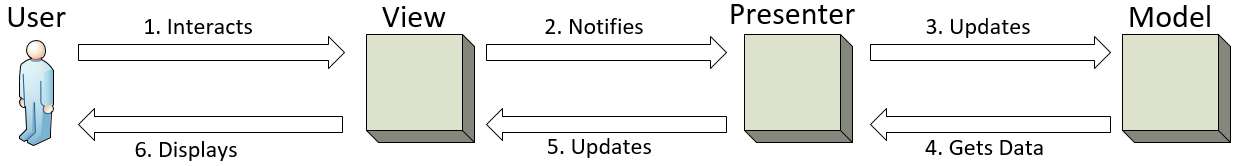
The state pattern can be interpreted as a strategy pattern which is able to switch the current strategy through invocations of methods defined in the pattern's interface.

All micro apps are represented as states in ref app, state pattern is used to switch between micro apps as part of user journey.

In Mobility Platform perspective, state pattern changes the state of the app depending on the user clicks. For example, if user clicks user registration, then the app state changes to user registration state and so on.

#### MVP Pattern

MVP is a type of software architectural pattern. MVP consists of 3 layers, namely Model, View, and Presenter. In this pattern, View receives the UI events and calls the Presenter as needed. Presenter is also responsible for updating the View with the new data generated by Model. The figure below shows the relation among the three layers.



##### Model

Model can be thought of as the interface to the data. Any part of the program, which needs some data to work on, must go through the interface or functions defined. Typically, model houses all the validation routines for the data submitted by the end user.

##### View

View, as the name implies, is the part where end user interacts. A program may have any number of views.

##### Presenter

Presenter acts as an intermediary to make the decoupling possible. All the business logic required for responding to a user event is written inside the Presenter layer. Typically the view only has the event handler and the logic to call the appropriate presenter functions, helping the person working on the view to concentrate upon designing the user interface without worrying about the code behind file. Presenter is also responsible for retrieving the requested data from the model and formats it so that the view can render it without any overhead.

Each presenter interacts with Flow manager to transition to different state or take necessary actions.

### uApp Development Rules

uApp framework defines standard set of interfaces which every micro app has to implement.

Any micro app developed should follow the guidelines as mentioned in uApp Framework guidelines document.

Please refer uApp framework design guidelines document for more details.

### Execution Architecture

#### Rules

##### General rules for developing with the platform

Solutions that develop on top of the platform shall comply with the rules below.

* The platform SHALL be used as a whole as far as applicable for the solution at hand
  + FrontEnds required for a solution use the Ref App supplied by the platform
  + All solutions SHALL include the OneBackEnd as a whole, not as individual services.
  + Connectivity components (BLE, WIFI) need to be included when devices with that connectivity technology are part of the solution
  + In specific cases with explicit agreement, individual components can be used. E.g. the UI Toolkit
* In case a solution needs a certain functionality it SHALL use the platform component that supports that functionality, if available.
  + Rationale is the drive for reuse, quality, harmonization of UX, concepts and behaviors.
  + This is a general rule that covers all artifacts such as infrastructure, connectivity, µServices, µApps, toolkits, SDKs.
  + This also holds for components that after mutual agreement have been harvested into the platform from that same solution.
* Solution specific µApps and µServices SHALL to comply to the rules defined in uApp framework document
  + This includes the mandatory use of AppInfra, UI Toolkit, uAppFWK.
* The latest platform version SHALL be adopted at least once a year or for any major new versions of the Mobile and Web Apps and new devices as part of the solution.
  + Rationale is to a) limit the number of platform versions to support b) maintain the harmonization and uniformity across the health continuum c) prevent buildup of technical debt.
* Mobile Apps and Devices SHALL log their version information.
  + Rationale is to be able to execute the deprecation process

##### Platform inclusion rules

Components can be part of the platform if

a) If there are common functionalities which caters to different business needs.

b) The component complies to the other architectural rules as stated below.

c) Actual inclusion and timing thereof will be determined by the governance.

##### Execution Parameters

###### Threading

A thread is a basic unit of CPU utilization, consisting of a program counter, a stack, and a set of registers, ( and a thread ID. ). Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others. Particularly true when one of the tasks may block, and it is desired to allow the other tasks to proceed without blocking.

**Benefits of Threading**:

There are four major categories of benefits to multi-threading:

* **Responsiveness** - One thread may provide rapid response while other threads are blocked or slowed down doing intensive calculations.
* **Resource** sharing - By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
* **Economy** - Creating and managing threads ( and context switches between them ) is much faster than performing the same tasks for processes.
* **Scalability**, i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be split amongst available processors. ( Note that single threaded processes can still benefit from multi-processor architectures when there are multiple processes contending for the CPU, i.e. when the load average is above some certain threshold. )
* Smoother Context Switching

**Challenges**:

* Security Issues: because of extensive sharing of resources between multiple threads.
* Threading has a real cost to the program (and the system) in terms of memory use and performance ( thread cancellation is also a costly affair)
* Difficulty of coding, testing and debugging
* The task of managing concurrency among threads is difficult and has the potential to introduce new problems into an application

**Things to consider**

* Do not block the UI thread
* Create threads judiciously
* Network calls and processing should be done on the worker thread
* Leave the priorities of the threads at their default values
* Ensure thread handles the exception
* Do not access the Android UI toolkit from outside the UI thread

##### Memory Management

**iOS & Swift:**

* Memory Management Issues
  + Freeing or over-writing data that is still in use. It causes memory corruption
  + Not freeing data that is no longer in use causes memory leaks
* How it happens
  + Retain cycles (use weak instead of nonatomic)
  + Unnecessary caching (Caching is ideal for storing frequently accessed objects, and we were not frequently accessing these images)
  + Not knowing what ARC handles in C
* Memory Management Rules
  + We own the objects we create, and we have to subsequently release them when they are no longer needed
  + Use Retain to gain ownership of an object that you did not create. You have to release these objects too when they are not needed.
  + Don't release the objects that you don't own.
* How to debug and avoid memory issues
  + Override the dealloc method
  + Manually create autorelease pools ( Could ignore in pure Swift)
  + Isolate possible problem areas
* Use Instrumentations to debug memory leaks

**Android:**

* How Your App Should Manage Memory
  + Use services sparingly
  + Release memory when your user interface becomes hidden
  + Release memory as memory becomes tight (use LRU principle)
  + Use Memory
    - Check how much memory you should use
    - Avoid wasting memory with bitmaps
    - Be aware of memory overhead
  + Use optimized data containers
  + Be careful with code abstractions
  + Avoid dependency injection frameworks
  + Be careful about using external libraries
  + Optimize overall performance
  + Use ProGuard to strip out any unneeded code
  + Use zipalign on your final APK - zipalign is an archive alignment tool that provides important optimization to Android application
  + Analyze your RAM usage
  + Use multiple processes
  + Analyze and re-think before using 3rd party library
    - ORM LITE – DATA access object DAO
    - Retrofit – time consuming in case of huge JSON parsing
* Use nano protobufs for serialized data

##### UI Performance

**iOS:**

* Do Not Block the Main Thread
* Caching – while fetching a remote resource, it is important that it doesn't fetch that same resources every time it needs to access it
* Lazy Loading - postpone the instantiation of an object until you need the object
* Measure Performance – Profile the application using the profiling tools
* Stress Testing (Load Testing)– Test the app with the right type of data and try to set up the real time environment. i.e if the app works with too much of data, ensure the app is tested for the same. Majorly app fails when it goes to real time load.

<https://www.raywenderlich.com/31166/25-ios-app-performance-tips-tricks>

**Android:**

* Improving Layout Performance
* Optimizing Layout Hierarchies
* Re-using Layouts with <include/>
* Loading Views On Demand
* Re-use & recycle resources wherever possible
  + Helps in - ListView Scrolling Smooth
* Do not create new objects, until and unless needed

##### Network Performance

A network operation costs the user time, and money

**iOS:**

* Using Power And Bandwidth Efficiently
  + Batch Your Transfers, and Idle Whenever Possible
  + Download the Smallest Resource Possible, and Cache Resources Locally
* Handling Network Problems Gracefully
  + Design for Variable Network Interface Availability
  + Design for Variable Network Speed
  + Design for High Latency
  + Test Under Various Conditions

**Tips for Efficient Networking:**

Implementing code to receive or transmit data across the network is one of the most power-intensive operations on a device. Minimizing the amount of time spent transmitting or receiving data helps improve battery life. To that end, you should consider the following tips when writing your network-related code:

* For protocols you control, define your data formats to be as compact as possible.
* Avoid using chatty protocols.
* Transmit data packets in bursts whenever you can.

##### Database Performance

* Design your database with caution
* Know what you should optimize
* Don’t select what you don’t need
  + Fetching of data thats not needed must be avoided
* Use Limits
  + SELECT title, excerpt, author FROM wp\_posts LIMIT 10;
* Avoid queries in Loop
  + Its like overloading your database with queries
* Use Join instead of subqueries
  + Join is faster than subqueries
* Be careful while using Wildcards
  + Wildcards are very useful because they can substitute for one or more characters when searching for data in a database. But be careful.
* Use UNION instead of or
  + The UNION statement allows you to combine the result sets of 2 or more select queries
* Use Indexes
  + They allow the database to find the requested information faster

##### Battery Performance

* Reduce Network Calls
  + Optimize API calls to reduce Network call, thereby reducing power consumption
* Alter the app's update rate by determining, and monitoring, the current battery level and changes in charging state
* Use best practices for Animation
  + Reduce the frame rate of animations
  + If heavy animations are involved, ensure CPU is in sleep state
  + Reduce the Size of the Bitmap Assets
  + Disable Anti-Aliasing when Drawing Scaled Bitmaps
  + Use bitmap filtering
* Detecting location- be Judicious in doing so

### Rules and Guidelines

1. Reference app is expected to be used as reference solution or an example proposition app.
2. Each proposition is expected to set up required backend data as per the components included in the app.

Example set up of service discovery, prx etc

<https://confluence.atlas.philips.com/display/MPA/What+a+Business+Needs+to+Setup+Before+Using+BaseApp>

1. The configurations and data input to all common components/micro apps should be fine tuned as per proposition needs after consuming reference app.
2. Reference app screens or user journey represents common use case and should be treated as an example however a propsotion can make use of designs and make relevant changes as per the requirement.

Please refer 3.3.5 for more details.

## Architecture Views

### System decomposition view

The following block diagram provides a high-level overview of the functional components in the ref app.

IAP

Product Registration

User Registration

Consumer care

Data Services

UIKIT

App Infra

uApp Framework

CommLib

The core functionality of Ref app is to provide a quick starting point for application development. It is a base application that pre-integrates all CDP2 common components.

#### uApp framework

Currently uApp framework is formed as a component which consists of standard set of interfaces for micro app development and also it contains Flow manager code.

##### uApp interface :

uApp interface provides a common interface to plugin any micro app in base app wrt the below,

* Common interface for performing init and launch of uapp.
* Unified way of handling action bar/Navigation bar
* Handling Back key
* Handling dependencies
* Handling Configuration
* Common interface or protocol for a micro app interface

##### Flow manager:

Flow manager is a programmable state machine which helps in transitioning from one state to another state in the app flow. Any flow element is termed as state.

The decision to move to next UI state in the flow is taken from JSON file named "AppFlow.json"

There are set of states in which an event can occur which leads to transition to new state based on few conditions.

Example : When you are in User registration state, you get logged in as an event and you would move to next state based on conditions as mentioned below.

Conditions are executed sequentially, if the first condition is met, app would move to the next state as defined in the flow otherwise iterate through next conditions.

An application need to define the app states, possible events in those states and conditions to move to next state.

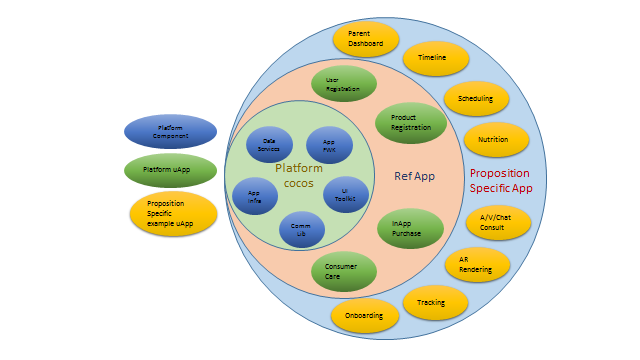
**Constraints:**

* To check for multiple conditions through json is not supported yet.
* Backward navigation through json is responsibility of propositions where in platform considers back as an event through standard event name called “back”.
* Creation of app flow via cloud is not tested yet though APIs are supportive.

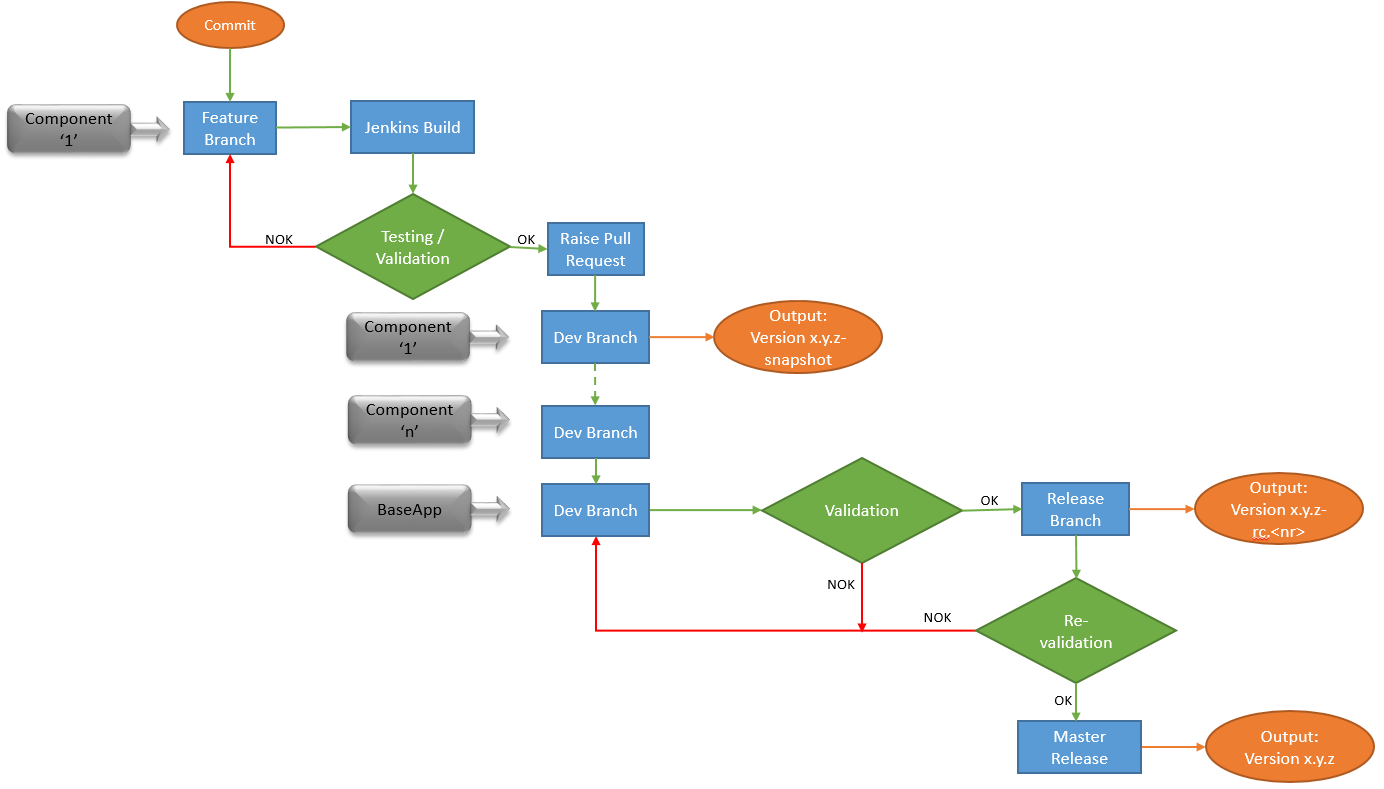
#### Common components and micro apps.

|  |  |
| --- | --- |
| Common component | Description |
| User Registration | The User Registration component provides a standard user experience for the user registration flow including social login. Base app integrates this component which abstracts the interaction towards the HSDP identity management service and various SDKs for social login and Janrain SDK. |
| InApp Purchase | IAP is the component that provides user interface to purchase products, services and accessories via mobile app. It provides the User Interface, connection to payment gateway and store. |
| Philips UI Kit | Philips UI Kit component provides the interface to create UI elements that are complaint with Philips design guidelines for Mobile Apps. |
| CommLIb | BlueLib handles the communication between App and Bluetooth device. The services and characteristics specific to device are implemented using BlueLib plugin.  Comm lib is a component that manages the app to device communication via wifi using dicomm protocol and also provides an abstraction over BlueLib APIs to communicate to Bluetooth device. |
| Consumer Care | The consumer care component is an off the shelf component providing consumer care functionality like customer care contact information, product information, product FAQs, service centers details, reviews and ratings etc. |
| Product Registration | The Product Registration component provides a standard user experience to register product under a registered user. |
| App Infra | App infra provides functionality that is common for most of the propositions and common components. It provides a base layer for optimal app development. App Infra provides logging , tagging , secure storage , service discovery and many other functions. |
| Data Services | It is developed as a common component that helps in syncing moments data between local data base and data core cloud. |

### Software View



### Process view



# Allocation of Quality Aspects

The concept of execution architecture is introduced on platform side to define performance criteria, NFRs etc.

Please refer 3.2.5 for more details.

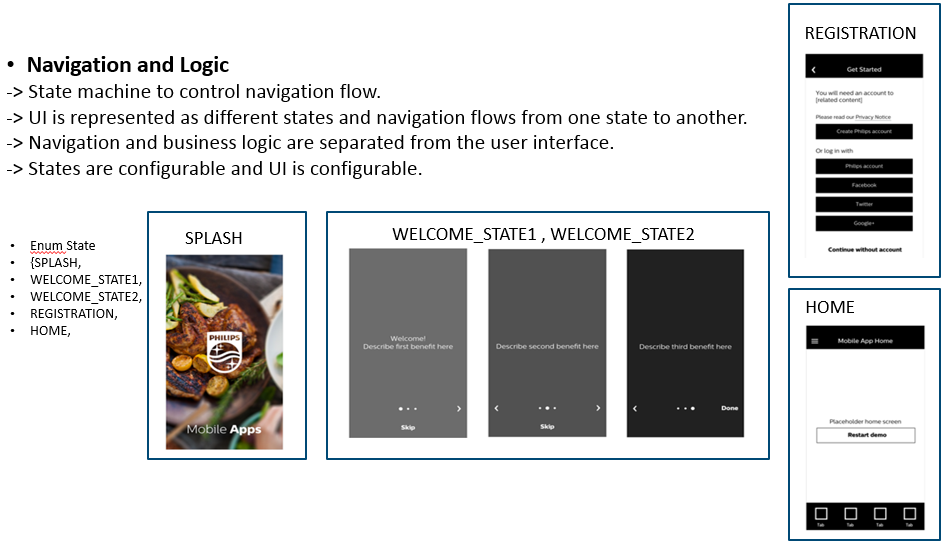
# Design Details

The User interface is build on the concept of micro app. The micro app boundaries are well defined.The microapps do not communicate with each other.

The flow manager manages all the flow in the app.Navigation across micro apps is based on the concept of state machine.

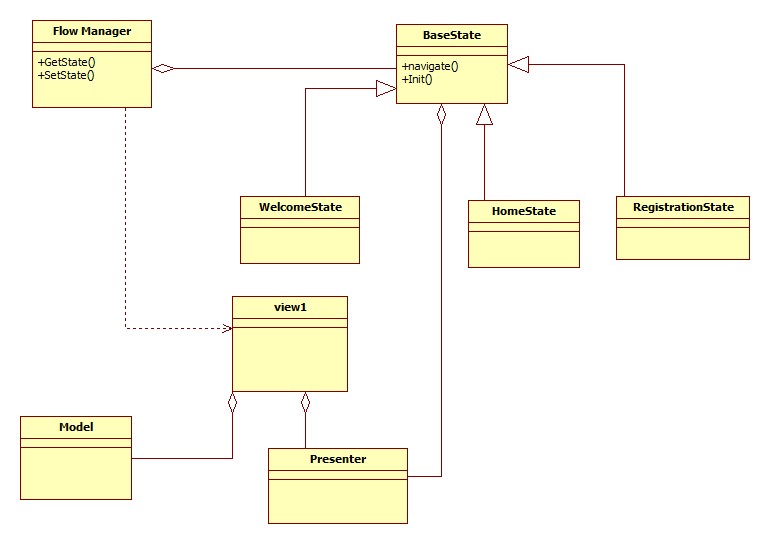
The flow manager creates the state and navigates the app to the next state as defined.

## Navigation Logic



1. A particular flow in the app is managed by the flow manger, A flow is a transition across the states.
2. BaseState is the base class for any state.
3. A state is an abstraction on top of standard micro app interface which helps in communicating to micro app.

## Configurable UI.



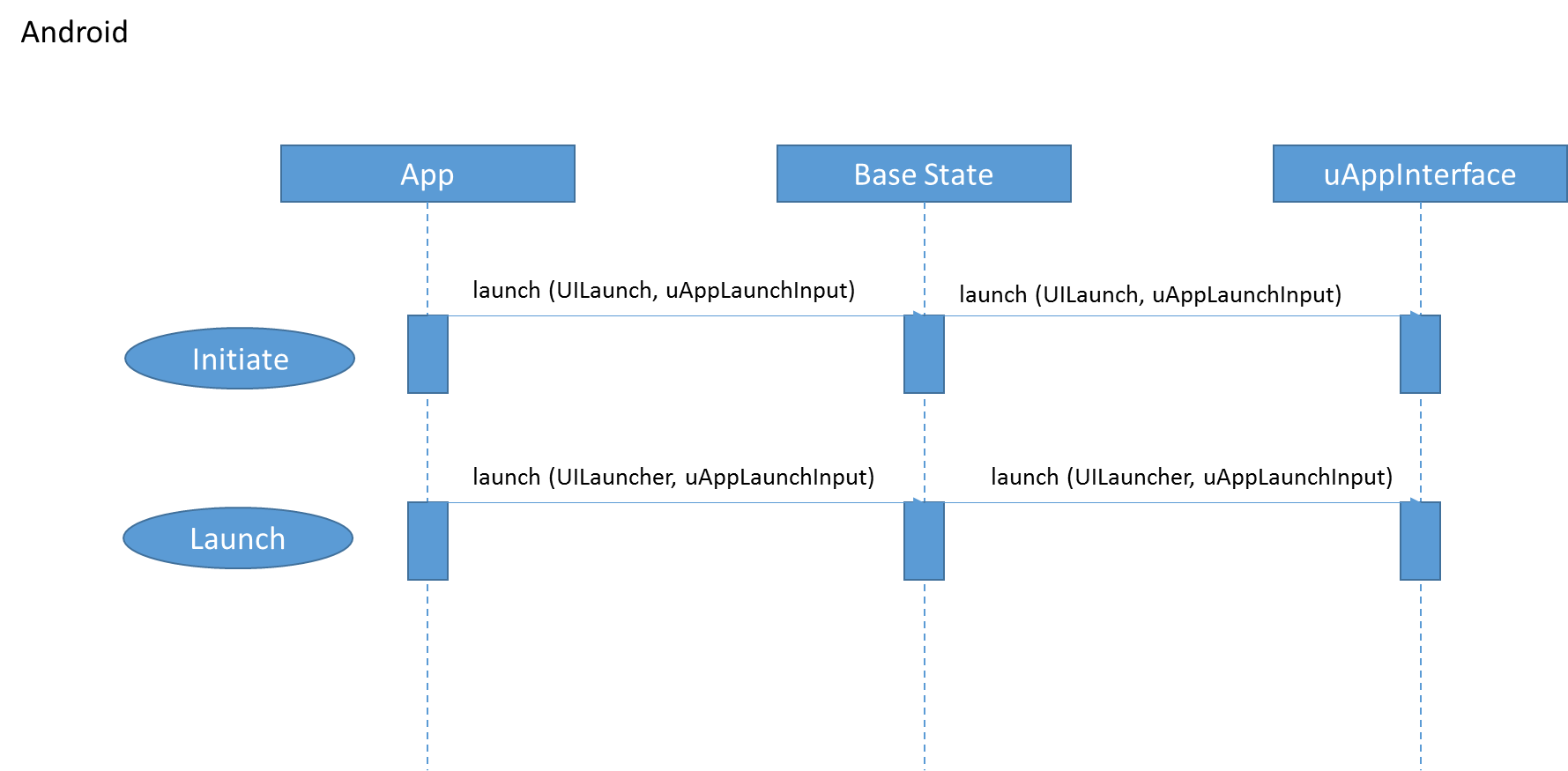
## External Interfaces

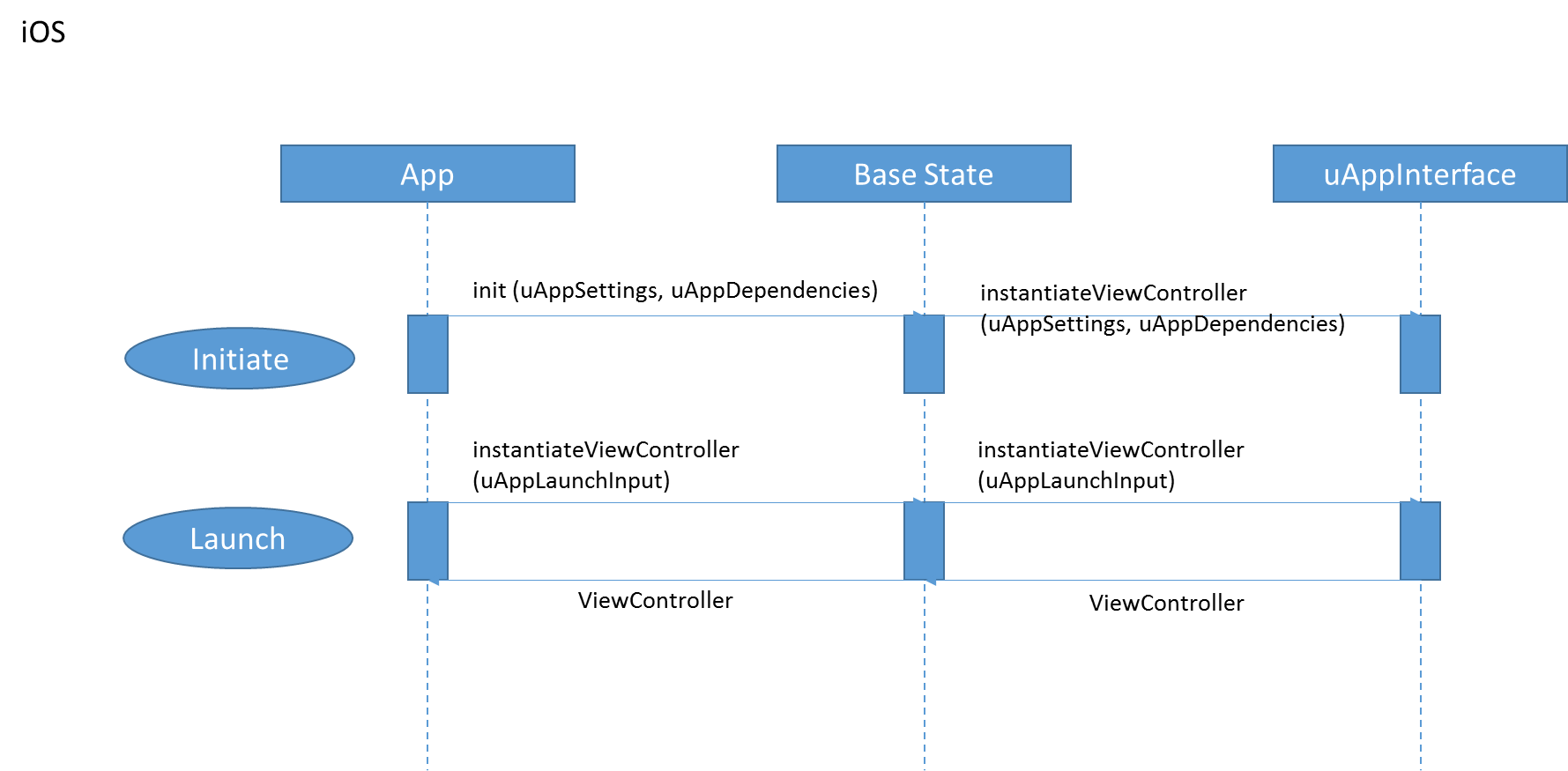
The external interface comes from individual components and reference app does not explicitly mandate any interfaces.

## Internal Interfaces

Reference app has following philosophy in terms of internal interfaces,

1. Any flow element in the app like settings, uApps is represented as state.
2. Interaction to any micro app will happen through a state class.
3. Each state class contains code related to a specific uApp or specific flow element.





# Revision History

| **Version** | **Date** | **Author** | **Description of Change** | **Reason for Change** |
| --- | --- | --- | --- | --- |
| 0.1 | 2016-06-29 | Ajay Das | First version |  |
| 0.2 | 2016-07-20 | Deepthi Shivakumar | Added micro app framework section | Introduced micro app framework |
| 0.3 | 2016-09-23 | Deepthi Shivakumar | Modified documents with latest updates.  Moved micro app section to different document. | Update needed for PI |
| 0.4 | 2016-11-12 | Deepthi Shivakumar | Addressed review comments and added more details about flow manager. | Update needed for PI |
|  |  |  |  |  |
| 0.5 | 2017-02-24 | Deepthi Shivakumar | Modified as per QMS 2.0  Template, referring whole content in terms of ref app instead of base app, detailed explaination on architectural elements. | Update needed for PI 17.1 |
| 1.0 | 2017-05-20 | Deepthi Shivakumar | Addressed few other comments , modifiedminor details in few sections. | Update needed for 17.2 |

# Approval

| **Name** | **Role / Function** | **Date** (YYYY-MON-DD) | **Signature** |
| --- | --- | --- | --- |
| Raymond Kloprogge | Chapter Architect | 2017-05-23 |  |
|  |  |  |  |