

WIT MSc Communications Software  
Architecture Description (AD)

WIT Design Patterns Assignment AD: Pacemaker Android Application

*<< Note: This Architecture Description (AD) template from the Open Group TOGAF™ 9.1 architecture framework has been adapted to the purposes of the WIT MSc module on IT Architecture Patterns.>>.*

Table of Contents

[A)Problem Description 6](#__RefHeading___Toc2122_729351093)

[A.1PROBLEM SCOPE 6](#__RefHeading___Toc2124_729351093)

[A.1.1PROBLEM SUMMARY 6](#__RefHeading__163_1468066642)

[A.1.2DOMAIN GLOSSARY 9](#__RefHeading___Toc2126_729351093)

[A.2STAKEHOLDERS CONCERNS 9](#__RefHeading__10_1468066642)

[A.3REQUIRED QUALITY PROPERTIES 9](#__RefHeading__18_1468066642)

[B)Solution Description 11](#__RefHeading__42_1468066642)

[B.1ARCHITECTURE VIEWS 11](#__RefHeading___Toc2128_729351093)

[B.2CONTEXT VIEW 12](#__RefHeading__44_1468066642)

[B.2.1VIEW INTENT 12](#__RefHeading__193_1468066642)

[B.2.2VIEW MODELLING ARTIFACTS 13](#__RefHeading___Toc2134_729351093)

[B.2.2.1SYSTEM CONTEXT MODEL 13](#__RefHeading___Toc2136_729351093)

[B.2.2.2SYSTEM FEATURES MODEL 19](#__RefHeading___Toc1991_1386314816)

[B.3BEHAVIORAL VIEW 20](#__RefHeading___Toc1993_1386314816)

[B.3.1VIEW INTENT 20](#__RefHeading__193_14680666421)

[B.3.2VIEW MODELLING ARTIFACTS 20](#__RefHeading___Toc2134_7293510931)

[B.3.2.1DATA PREPARATION MODEL 20](#__RefHeading___Toc2136_7293510931)

[B.3.2.2DASHBOARD RENDERING MODEL 23](#__RefHeading___Toc1995_1386314816)

[B.4INFORMATION VIEW 30](#__RefHeading___Toc1997_1386314816)

[B.4.1VIEW INTENT 30](#__RefHeading__193_146806664211)

[B.4.2VIEW MODELLING ARTIFACTS 34](#__RefHeading___Toc2134_72935109311)

[B.4.2.1BATCH ANALYTICS DATA FLOW MODEL 34](#__RefHeading___Toc2136_72935109311)

[B.4.2.2DASHBOARD RENDERING DATA FLOW MODEL 36](#__RefHeading___Toc1999_1386314816)

[B.5FUNCTIONAL VIEW 38](#__RefHeading___Toc2001_1386314816)

[B.5.1VIEW INTENT 38](#__RefHeading__193_1468066642111)

[B.5.2VIEW MODELLING ARTIFACTS 38](#__RefHeading___Toc2134_729351093111)

[B.5.2.1INDIRECTION LAYERS MODEL 38](#__RefHeading___Toc2136_729351093111)

[B.5.2.2DATA PROVISIONING MODEL 40](#__RefHeading___Toc2003_1386314816)

[B.6ARCHITECTURE PERSPECTIVES 42](#__RefHeading___Toc2149_1386314816)

[B.6.1PERFORMANCE PERSPECTIVE 42](#__RefHeading___Toc2151_1386314816)

[B.6.1.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 42](#__RefHeading___Toc2153_1386314816)

[B.6.2SCALABILITY PERSPECTIVE 42](#__RefHeading___Toc2155_1386314816)

[B.6.2.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 42](#__RefHeading___Toc2157_1386314816)

[B.6.3TESTABILITY PERSPECTIVE 43](#__RefHeading___Toc2159_1386314816)

[B.6.3.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 43](#__RefHeading___Toc2161_1386314816)

[B.6.4FLEXIBILITY PERSPECTIVE 43](#__RefHeading___Toc2163_1386314816)

[B.6.4.1RECORDING of DESIGN DECISIONS / TRADE-OFFS 43](#__RefHeading__110_1666809543)

[B.7VIEW MAPPINGS 43](#__RefHeading___Toc2140_729351093)

[B.7.1For each VIEW MAPPING 43](#__RefHeading___Toc2142_729351093)

[C)References 44](#__RefHeading___Toc2146_729351093)

Document Information

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Name:** | Pacemaker Android Application | | |
| **Prepared By:** | Colm Carew - 20053766 | **Document Version No:** | 1.0 |
| **Title:** | Architecture Description (AD) | **Document Version Date:** | 2016-05-02 |
| **Reviewed By:** | Colm Carew | **Review Date:** | 2016-05-03 |

Distribution List

| **From** | **Date** | **Phone/Fax/Email** |
| --- | --- | --- |
| Colm Carew | 2016-05-02 | N/A |
|  |  |  |

| **To** | **Action\*** | **Due Date** | **Phone/Fax/Email** |
| --- | --- | --- | --- |
| Siobhan Drohan | Review | 2016-05-12 | sdrohan@wit.ie |
| Eamon De Leastar | Review | 2016-05-12 | edealeastar@wit.ie |
| Pierre Peclier | Review | 2016-05-12 | ppeclier@wit.ie |
|  |  |  |  |

\* Action Types: Approve, Review, Inform, File, Action Required, Attend Meeting, Other (please specify)

Document Version History

| **Version Number** | **Version Date** | **Revised By** | **Description** | **Filename** |
| --- | --- | --- | --- | --- |
| **1.0** | May. 02nd | Colm Carew | This document files some of the Architecture Views describing the Android and Play Application Assignment of the Design Patterns module: Pacemaker Android Application. | pacemaker\_architecture.archimate |
|  |  |  |  |  |

Document Overview

The following document shows the Architecture Views of the Pacemaker Android Application for the Design Patterns Module Assignment on Architecture Styles & Patterns into a TOGAF Architecture Description.

# Problem Description

## PROBLEM SCOPE

### PROBLEM SUMMARY

|  |  |
| --- | --- |
| **Project Name** | Pacemaker Android Application |
| **Industry Domain** | Activity Tracking |
| **Problem Description** | A Play Application was created in the Agile Software Development Module. This assessment seeks to create an Android Application which interfaces with the Play Application using the REST API.  The Android Application will mirror the Play Application as closely as possible as well as add some features specific to the Android Application.  The Android Application should:   1. Track and Log activities and basic stats 2. Have progress reports based on completed activities 3. Connect with friends and compare workouts with friends 4. Prescribe workouts based on completed activities   The Android Application should also exhibit and number of code design patterns. |

### DOMAIN GLOSSARY

|  |  |
| --- | --- |
| **User** | A registered Pacemaker User can track their activities, connect with friends, compare workouts and generate prescribes workouts based on their completed activities. The User can log in via colmcarew.com or via the Android Application. Comparing and prescribing workouts are features solely for the Android Application. |
| **Activities** | Activities are exercised based activities which hold data such as date, duration and distance. These figures are then used for prescribing workouts and progress reports. |

## STAKEHOLDERS CONCERNS

|  |  |
| --- | --- |
| **Initiative Sponsors** | Pierre Peclier, Eamon De Leastar, Siobhan Drohan |
| **Project Development Team** | Colm Carew |
| **System Administrator** | Colm Carew |

## REQUIRED QUALITY PROPERTIES

|  |  |
| --- | --- |
| **Performance** | System must have acceptable performance and response time for requests. Proximity of components should remain close to not lose performance due to communication overhead. |
| **Usability** | The Android Application must be simple to understand and use.  The user should be able to navigate through all the Android Application’s features without any impediments. |
| **Executability** | The Play Application should be easily deployable to another server without large issues (true as it is ran from Jar files).  The Play Application should run as cheaply as possibly due to limited resources (true as the Play Application and database are ran from a single Raspberry Pi). |
| **Portability** | The system should be relatively easily built upon for a different platform such as iOS (true due to the heavy implementation of REST) |

# Solution Description

Below are the key characteristics of the solution design.

## ARCHITECTURE VIEWS

|  |  |  |
| --- | --- | --- |
| **VIEW NAME** | **PURPOSE** | **TARGET STAKEHOLDER(s)** |
| **CONTEXT View** | The purpose of this view is to outline the main elements at play in the design of the Pacemaker dashboard:  1.What is the purpose of the system  2.What are the systems features | Data Analyst, CIO |
| **BEHAVIORAL View** | List processing flows at play for the rendering Pacemaker's dashboard. The purpose of this View is to outline:  1.How data gets collected and prepared for storing  2.How analytics data is served to users | CIO  Data Analysts |
| **INFORMATION STRUCTURE View** | Describes how the Pacemaker system stores and manages data.   1. Analytics Data Flow - how the application processes and calculated data 2. Rendering Data Flow – how data is stored and retrieved | Data Analysts  Developers |
| **FUNCTIONAL View** | This specifies the key components of the Pacemaker system and their relationships with one another. | Developers |
| **INFRASTRUCTURE View** | The purpose of this view is to outline the infrastructure behind the design such as servers, databases, applications, how they connect to one another and how the user can connect to the applications. | System Administrators |

## CONTEXT VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **SYSTEM CONTEXT Model** | The purpose of this model is to show the solution in terms of system components. |
| **SYSTEM FEATURES Model** | The purpose of this model is to show the solution in terms of it’s features. |

### VIEW MODELLING ARTIFACTS

#### SYSTEM CONTEXT MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Context View Type** | To illustrate the relationship between the solution and the environment. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **System Context Modelling** | This model shows the Pacemaker Dashboard capabilities in the centre of the model.  All the user data is stored and collected from the MySQL database.  Analytics data is calculated at run time and is not stored. Data used to calculated the analytics are the finished activities which are stored in the database. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Data-driven Style: Agile Data Processing** | User Analytics data is calculated at run time and is not persisted to any database. Data regarding activities is pulled from the database when the user requests analytics calculations so all calculations are near real time. |
| **Domain Pattern: Shared Database** | User data, activities and friends are all stored within the same database. Data can be pulled or pushed from external applications into this database via REST with the Pacemaker Play Application. |

#### SYSTEM FEATURES MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Context View Type** | The purpose of this view is to outline the main features of the Android Application. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **System Feature Modeling** | This model shows that the Application is composed of one large component, the Pacemaker Dashboard.  This model shows how features of the solution are decomposed and allocated within the main components.  Most of the features require some data being pulled and manipulated from the database or data being edited/created. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Data-driven Style: Agile Data Processing** | User Analytics data is calculated at run time and is not persisted to any database. Data regarding activities is pulled from the database when the user requests analytics calculations so all calculations are near real time. |
| **Domain Pattern: Shared Database** | User data, activities and friends are all stored within the same database. Data can be pulled or pushed from external applications into this database via REST with the Pacemaker Play Application. |

## BEHAVIORAL VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **Data Persistence** | This model describes the data persistence process. The process is triggered every time a User edits/create data(e.g. a new activity, a modification of an older activity) in the Android Application.  The flow describes how all data is created, edited and deleted in this Pacemaker Android solution. |
| **Data Rendering** | This model describes the type of tasks the Pacemaker Dashboard performs to fetch the activity data from the Pacemaker website (REST communication), and transform the data to render it on the Android screen. |

### VIEW MODELLING ARTIFACTS

#### DATA PERSISTENCE MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Behavioral View Type** | The purpose of this view is to show communication mechanisms required to coordinate operations between functional elements.  It also outlines triggers, events, and flow-logic.. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Process Flow Modeling** | This model shows how a user request in the Android Application for editing/creating data persists to the database through REST requests. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Service Registry Pattern** | The Android Application needs to determine the location of the Pacemaker website in order for it to to send REST requests. |

#### DATA RENDERING MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Behavioral View Type** | To outline the process by which the activity and friend data is fetched and rendered in the Android Application.  It also outlines the steps that manipulate data before it is displayed to the user. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Process Flow Modeling** | This model shows the steps involved in activity and user data being obtained from the Pacemaker website via REST. The data is then processed via the Android Application and displayed on screen for the user to view. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **MVC Pattern** | Model-View-Controller pattern to render Android Activity pages based on the data collected and manipulated from the user activities and friends. The model for this pattern is the data (the user or the activity), the controller is the logic behind processing and manipulating the data and the view is the Android Activity the user sees. |

## INFORMATION VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **Analytics data flow** | This model outlines the components that transform data pulled from the web application.  The data flow described in this model is a similar to an ETL model as the data is extracted from the database (REST from the web application), transformed via domain logic in the Android Application and finally loaded onto the users screen. |
| **Rendering data flow** | This model presents the flow of requests and responses of objects throughout application components. It is triggered by a user performing an action in the Android Application which causes data to be requested via REST. |

### VIEW MODELLING ARTIFACTS

#### ANALYTICS DATA FLOW MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Information Services View Type** | Describes how the architecture stores data and define any significant data structures used within the system.  It also shows the data exchange between functional elements |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **ETL Orchestration Modelling** | To describe the roles of application components in the extract, transform and load phase of data. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Pipes & Filters Pattern** | IThe purpose of this pattern is to implement a message processing mechanism which is platform independent. Thus making it re-usable for future views.  IT also identifies the type of transformation functions and data objects involved in the input and output of each pipe and filter. |

#### RENDERING DATA FLOW MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Information Services View Type** | Describes how the architecture stores data and define any significant data structures used within the system.  It also shows the data exchange between functional elements |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Data Flow Modelling** | This model shows the flow of data when the Android Application makes a REST request to the Play Web Application. |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Synchronous Point to Point Processing pattern** | In this case this pattern sends a message from the Android Application to the one receiver which is currently the Raspberry Pi running the Pacemaker Web Application. There is only one receiver for now so only one receiver can receive the message. |

## FUNCTIONAL VIEW

### VIEW INTENT

|  |  |
| --- | --- |
| **MODEL NAME** | **PURPOSE** |
| **Ports and Adapters** | To outline the structure of the Dashboard Application (all components included) and identify coupling risks. To describe logical groupings of key components collaborating to achieve a similar function. |

### VIEW MODELLING ARTIFACTS

#### PORTS AND ADAPTERS MODEL

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **Functional View Type** | To describe the composition relations between components collaborating for the specific purpose of (1,) fetching data (i.e. Data Access Objects - DAO), (2.) aggregating data into Transfer Objects (TO) propagated back up to the Dashboard Client UI. |
| **MODEL KIND USED** | **JUSTIFICATION / INTENT** |
| **Application Structure Modeling** | Used in this situation to identify coupling between layers of components of a same nature. Also used to identify servicing points and data contracts reducing coupling (typically Application Services, API, etc.) |
| **STYLE/PATTERNS USED** | **JUSTIFICATION / INTENT** |
| **Layered Style – Relaxed Variant (includes 2 API service points)** | The variant of the Layared pattern used in the Analytics Dashboard architecture aims to primarily isolate domain model components – but also to eliminate any dependency between data infrastructure concerns and user interface experience, so to prepare for Mobile UI Clients with a minimum of rework. |

## ARCHITECTURE PERSPECTIVES

### PERFORMANCE PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **Need to prevent session time-out for users of the Analytics Dashboard.**  **Response time of Analytics Dashboard must be less than 8 seconds for 5,000 concurrent users.** | **Decision**: A Data Analytics View will not be generated on-demand when an ELITE User connects to a dashboard.  Generating data analytics on-demand is technically feasible but this architectural options doesn’t scale above 2,500 concurrent users given the current infrastructure in place at Runkeeper.  **Trade-off:** Data Analytics will be generated asynchronously by a background process named (Batch Analytics). |
| **#2** | **Some ELITE Users will want to see their stats immediately after upload. At that point, the analytics job will not be yet complete, and the dashboard will not reflect reality.** | According to current usage statistics of simpler Runkeeper dashboards, 95% of Users are not checking their stats immediately after ending an Activity.  However 5% of ELITE Users may want to see their new activity stats immediately reflected after uploading their Health or Activity data into Runkeeper.  **Trade-off**: For these users, an indicator will be displayed on the Dashboard page to notify that their last data upload is in progress. *Eventual Consistency* in this case is a satisfactory option for these Customers. |
| **#3** | **Performance MUST BE independent of processing volume.**  **Throughput of data MUST BE predicatble.** | **Ref to #1** |
| **<#>** |  |  |

### SCALABILITY PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **Cost of required processing cycles and memory must be predictable, function of the number of concurrent users.**  **Starting from 5,000 concurrent users, the supporting infrastructure MUST dynamically scale up 100,000 concurrent users by means of infrastructure configuration.** | **Decision**: No net-new scalability strategies will be introduced for the Rendering of the Web Data Analytics Dashboard.  The Dashboard UI is based on a classic Web architecture sourcing data from a shared database, and as such follows the same scalability strategies as any other form of dynamic web-content currently generated by Runkeeper (using HAProxy, Load Balancer and Caching mechanisms).  Runkeeper has the infrastructure and tools in place to deal with Web workload going way beyong 5,000 users.  **Decision:** A new strategy will be defined however to Scale-out the resource-intensive generation of Data Analytics Views.  The Asynchronous Batch Analytics architectural option allows for a better exploitation of server resources (CPU and memory), minimizing the use of the same components and communication paths.  Batch Analytics implementations can be optimzed for repeated processing. Contention can be reduced via replication of ETL jobs.  **Trade-off**: Data processing processing jobs will be distributed over time instead of kick-off on-demand. Some analytics jobs will be prioritized for Users who are logged-in and are waiting to see results immediately. Other jobs will be queued for Users that are not connected. |
| **<#>** |  |  |

### TESTABILITY PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **Beyond testing the UI features of the Analytics Dashboard, the architecture MUST allow of load testing so monitoring of service levels of the application can be automated.** | **Decision:** Given above decisions and trade-offs for the Performance and Scalabilty Perspectives, it makes sense to automate Load Testing for Batch Analytics processes since it is the most heavily loaded component of the architecture.  **Decision**: A “load test” batch analytics process creating “dummy” analytics views will be consistently running on Runkeeper’s servers, exercising the server components at all times.  The end-to-end duration of the job will be constently controlled as a Key Performance Indicator of the architecture.  When / if the “load test” job falls below a certain duration window not meeting SLA requirements, a notification will be dispatched to System Admnistrators to adjust server configurations.  **Trade-off**: A new ETL Server will be acquired and act as a distribution node for other data processsing ETL servers.  At a minimum, 4 servers will compose the backend of Runkeeper data analytics (i.e. 1 Main Node, 3 Slaves).  Steps for scaling out further from there-on are, by order of preference: (1.) CPU and Memory additions to ETL server, (2.) Insertion of new node. |
| **<#>** |  |  |

### FLEXIBILITY PERSPECTIVE

#### RECORDING of DESIGN DECISIONS / TRADE-OFFS

|  |  |  |
| --- | --- | --- |
| .**ID** | **DECISION ITEM** | **DECISION MADE** |
| **#1** | **The proposed architecture must be extensible to an Android/IOS version of the Analytics Dashboard with minimum rewrite of the core system logic.** | **Decision**: A Layered Architecture will abstract the presentation layer from how/where User data analytics views are stored.  A data provider component will act as a Facade (gateway) to access Analytics Views from any future Client UI.  **Trade-off**: No Service API will be defined for the first revision of the Analytics Dashboard.  The Web UI Pages will exclusively use Transfer Objects passed via Server-Side Scripting mechanisms (i.e. Web MVC).  The Web UI will not use any Service API. The reason for this is that, since a Facade component acts as a gatway to access Analytics Views, it will be relatively easy to create an API Service sitting on he top of it (i.e. making use of it).  Domain Logic will not change, only the Data Provider, offering Transfer Objects passed via REST to the Mobile UI. |
| **<#>** |  |  |

## VIEW MAPPINGS

### For each VIEW MAPPING

|  |
| --- |
| Not covered in this example AD. |

|  |  |
| --- | --- |
| **VIEWPOINT USED** | **JUSTIFICATION / INTENT** |
| **<<VIEWPOINT NAME>>** | <<DESCRIPTION>> |

# References