Dynamic Performance Framework

Iris

Status Report (Semester 2)

Dean Gaffney

20067423 Panel 3

Supervisor: Dr. Kieran Murphy

Second Reader: David Drohan

BSc (Hons) in Entertainment Systems

Contents

List of Tables

List of Figures

CHAPTER 1

Abstract

The aim of this project is the design a full implementation of a system for application performance monitoring. The proposed system, has the working title, Iris.

On completion, Iris will provide users with a dynamic performance framework which will allow users to fully customise and centralise their application performance monitoring. This will be achieved through a web interface where a user can specify a schema for a specific application they wish to monitor. Once a schema has been set up, a REST endpoint will be generated for the application. This endpoint will allow a user to send their monitoring data from their desired application to the framework in the form of JSON (matching the specified schema). Iris will also contain features which allow a user to monitor and analyse incoming data, using an intelligent, fully customisable graph and dashboard builder. Iris will then visualise any received data in real time using to the appropriate dashboards using websockets. Iris will come with some out of the box scripts/applications that users can use to monitor typical tasks such as JVM (Java Virtual Machine) performance, Linux OS System Performance.

CHAPTER 2

Introduction

2.1 Motivation for Iris

The motivation for this project comes from database and system performance issues that Onaware[1] has experienced in recent projects. It is often the case that they must deal with large amounts of identity data being aggregated into a third party system called 'IIQ'.

Onaware has faced major issues with aggregating data in the past, in some cases it was taking up to five days, and sometimes they would fail halfway through meaning aggregations would have to be restarted, due to the amount of software involved it is hard to pinpoint what software is causing the issue.

In one such instance of aggregating data issues several attempts were made to rectify the performance issue such as optimising sql queries, increasing ram, multi threading tasks and increasing disk space, none of which worked. Due to the performance issue the IIQ instance became unusable so debugging the issue was not possible from inside the application and log files became so big that text editors would crash when trying to open them. In this case the issue turned out to be a customer putting size constraints on the database storing the aggregated data. While monitoring would not prevent such a mistake it would have reduced the time needed to locate the issue.

In response to difficulties in identifying performance issues Onaware have tried to monitor specific application elements. The aim at the time was to try and combine SQL, JVM and Operating System scripts to track the performance of the tools, however this approach is not very scalable and it would need to be reconfigured for future projects.

Iris is an attempt to solve this problem. Iris will allow a user create a new application monitor with little effort using a web interface, give the user a REST endpoint specific to the application for their scripts to target their data, and allow a user to monitor the data in real time using graphs and dashboards. The aim is to make the framework as flexible as possible and not

[1] Onaware is an international company that specialises in IAM (Identity and Access Management) and has offices with 20 staff in Waterford. More information on Onaware can be found at https://onaware.com.

specific to the issue Onaware faced, meaning a user can monitor any data they want from any application they want all they must do is send their data to a REST endpoint.

Users of Iris will consist of Onaware developers who will be monitoring IAM project data and generic tools which may be released to clients at a later time.

2.2 Features List

2.2.1 Types of data

numerical, categorical and textural

CHAPTER 3

Implementation

During semester 1 a number of implementation were designed and tested. As a result the implementation described in semester 1 required not further changes in thus sesmter,. Hence, this section is fundamentally unchanged from that in semester 1 report Distinguish between subparts that were implemented (and so their overview here is similar to that in semester 1 report) and subparts that were implemented during this semester (and so the overview here is new). In this chapter the implementation of Iris is summarised. The components completed in semester were described in the Semester 1 report and that summary is reproduced here (with minor modification) for completeness. The subsections dealing with components impleneted in semester 2 is fundmental new. Table REF lists the components adn when they were completed.

Component	When prototyped	When completed
code framework/services	Semester 1	Semester 1
schemas		
dashboards		
aggregiation builder		
data sources		

3.1 Core Framework/Services

3.2 Schemas

3.3 Dashboards

3.4 Aggregation Builder

3.5 Data Sources

CHAPTER 4

Deployment, Testing and Evaluation

4.1 Deployment

4.1.1 AWS

4.1.2 Jetty

4.1.3 Gradle

4.1.4 WAR

4.1.5 MySql GORM Mapper

4.2 Testing and Evaluation - Data Sources

4.2.1 Introduction

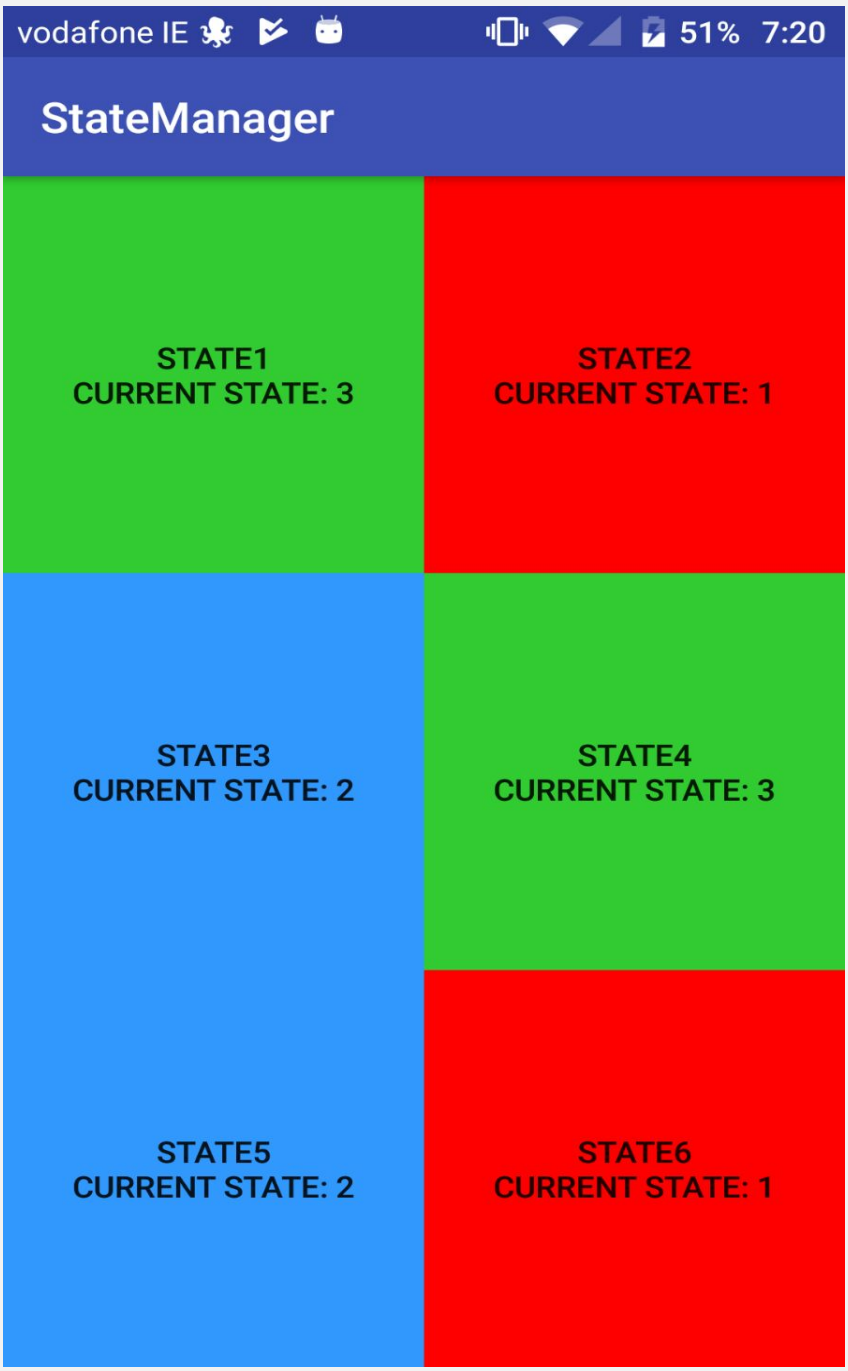
A number of data source have been implemented to demonstrate the flexibility of Iris and to test the aggregation. Each of the following sections describe a data source and its unique features.

4.2.2 Android App

4.2.2.1 Description

The android application demonstrates how Iris handles state based data. The application is very simple, but demonstrates how simple it is to monitor an application's state through Iris. The application is simply a screen consisting of six tiles. Each tile represents a different state, each state has three colours and three numerical values linked to the states and colours. The

android application allows the user to change the states of these tiles, which then changes the state of the tile colour and value. When a user is satisfied with the states they wish to send to Iris they simply tap the screen. This results in a JSON object being sent to Iris consisting of the current numerical value for each state tile i.e the current state of the tiles.



A screenshot of the android agent and its state based tiles.

Figure 4.1 – Android agent for Iris

4.2.2.2 Linkage with Iris

The android application is linked to Iris through a unique REST endpoint specific to the android application. All of the data being sent from the android application is sent to this endpoint. Once the data enters Iris, Iris will run through it's logic for checking for dashboards and charts

associated with the schema and send the data through to the charts. In this case the charts are state based and will update according to the state values that are given to them.

A screenshot of the android agent dashboard in Iris and its state based tiles.

Figure 4.2 – Android agent dashboard in Iris

4.2.3 Raspberry Pi

```
#!/usr/bin/python

import requests, json
from coinmarketcap import Market

headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}

agentData = {'name': 'crypto_agent'}

urlResp = requests.post('http://ec2-52-16-53-220.eu-west-1.compute.'+
'amazonaws.com:8080/iris/schema/getAgentUrl',
data=json.dumps(agentData), headers=headers)

endpoint = urlResp.json()['url']

wanted_keys = ['price_usd', 'price_eur','name',
               'percent_change_24h', 'rank']

def extract(data):
    return dict((k, data[k]) for k in wanted_keys if k in data)

coinmarketcap = Market()

crypto_currencies = coinmarketcap.ticker(limit=4, convert='EUR')

filterd_currencies = list(map(extract, crypto_currencies))

for currency in filterd_currencies:
    resp = requests.post(endpoint,
                         data=json.dumps(currency),
                         headers=headers)
    print resp.json()
```

Figure 4.3 – iris-crypto-rates.py soure code

4.2.3.1 Description

What's so special about this data source?

4.2.3.2 Linkage with Iris

Resulting dashboard (some images)

4.2.4 Nodejs

4.2.4.1 Description

What's so special about this data source?

4.2.4.2 Linkage with Iris

Resulting dashboard (some images)

4.2.5 Selenium

4.2.5.1 Description

What's so special about this data source?

4.2.5.2 Linkage with Iris

Resulting dashboard (some images)

4.2.6 MySQL

4.2.6.1 Description

What's so special about this data source?

4.2.6.2 Linkage with Iris

Resulting dashboard (some images)

CHAPTER 5

Conclusions

Bibliography

Docs.spring.io (2017). *STOMP Support*. URL: https://docs.spring.io/spring-integration/reference/html/stomp.html (visited on 11/14/2017).

Elastic.co (2017a). *Aggregations*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations.html (visited on 11/04/2017).

– (2017b). *Basic Concepts*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html (visited on 11/04/2017).

– (2017c). *Mapping*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html (visited on 11/04/2017).

Logz.io (2017). *Grafana vs. Kibana: The Key Differences to Know*. URL: https://logz.io/blog/grafana-vs-kibana/ (visited on 11/16/2017).

Niederwieser, Peter T. (2017). *Introduction*. URL: http://spockframework.org/spock/docs/1.1/introduction.html (visited on 11/11/2017).

Appendices