BILKENT UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

SENIOR DESIGN PROJECT

High Level Design Report

02.05.2016

Horus

Group Members

Ali İlteriş Tabak
Hakan Kılıç
Ömer Eren
Süleyman Can Özülkü
Yiğitcan Kaya

Supervisor

İbrahim Körpeoğlu

Jury Members

Özgür Ulusoy, Selim Aksoy

TABLE OF CONTENTS

1.0 Introduction

1.1 Purpose of the System

The development and adoption of Internet of Things is a critical element of data driven decision making. However, most of the technology to capture and track sensor measurements are developed and installed aren't connected to the internet and can only be seen by measurement panels. Real-time tracking of the data generated by legacy systems is impossible without changing whole system. In this project, we propose an interim technology to track legacy control panels without adding big overhead by using a simple camera and image processing techniques that will bring power of the cloud and big data analytics to help our customers with the analysis of the data they accumulated from their sensor screens.

1.2 Design Goals

- **Reliability (Fault Tolerance):** Since Horus will be applied to the sensitive production lines it needs to be reliable and must present the user data in an accurate way, without losing the data and system integrity. At minimum, the system must provide the necessary warnings in case the reliability is compromised in case of unexpected problems (such as power outages), in order to make the user and system admins take necessary cautions to restore the reliability.

- **Adaptability (Flexibility):** Since Horus will operate on various production lines, sensitive systems, and so on; it needs to be adaptable to such various environments. The sensors

Horus will need to operate on, are very diverse, thus it needs to reliable handle those sensors and their data. For example, it may be applied to an outdoor central of an electric distribution company, or an indoor chemical factory machine, thus it needs to be flexible and should be adapted for indoor, outdoor use on different types of sensors and different types of power sources. (Battery and plug)

- **User friendliness:** The end users of Horus are not necessarily technical users, they might be factory employees that just needs to reliably monitor and analyze their sensitive sensor data. Thus, user interface of Horus must be simple and clean that is suitable for non-technical users, also it should provide its functionality, such as data analysis functionality, on an easy-to-use interface.

- **Scalability:** Since the Horus will be a data-heavy application, it needs to be able to handle such data volume from multiple end-user appliances (devices). Also, in Horus, each user might have multiple appliances as most of the production lines consist of multiple it also needs to be scalable from a single user perspective.

- **Security:** Since Horus aims to handle sensitive data, it need to use robust cryptographic tools for client-server authentication (SSL/TLS), data encryption (AES), secure data storage, secure server back-end and on.

1.3 Definition, Acronyms and Abbreviations

SSL/TLS: (Secure socket layers / Transport layer security) An asymmetric cryptographic protocol that provides authentication, privacy and reliability to the communication between two parties that makes use of public key cryptography system.

AES: (Advanced Encryption Standard) A safe, state-of-the-art symmetric cryptographic block cipher algorithm that provides secure, single keyed encryption of data.

Beaglebone Black: BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists.

2.0 Current Technologies

Because of the sensor and gps tracking equipment prices there have not been many startups that focused on sensors prior to 2014. However, there is a startup called **Helium**[1]. Their main use case is to provide an all included sensor pack to place wherever the client wants. However, problem with providing sensor pack is that you have to replace your existing tracking systems with new sensors, and also specific sensors wanted by client may not be provided within the given pack. However, our project ensures that clients do not need to replace their legacy systems and deployment can be made where the existing tracking system lies.

3.0 Proposed Architecture

3.1 Overview

In this section we will try to demonstrate architecture of our system by giving architectural styles that will be used, subsystem decomposition, hardware/software mapping, persistent data management, access control - security, global software control and lastly boundary conditions.

3.2 Architectural Styles

Horus will be based on different architectural styles that will be used in architecture of the system.

3.2.1 Microservice

Microservice architecture contains many services which are small, independent processes that communicate with each other to form complex applications. These services are highly decoupled and facilitating a modular approach to system-building [2]. The reason behind choosing this architectural pattern is that the architectural pattern provides us implementing of services with different languages and on different hardwares. Another pro of the pattern is that architecture is symmetrical rather than hierarchical that provides flexibility.

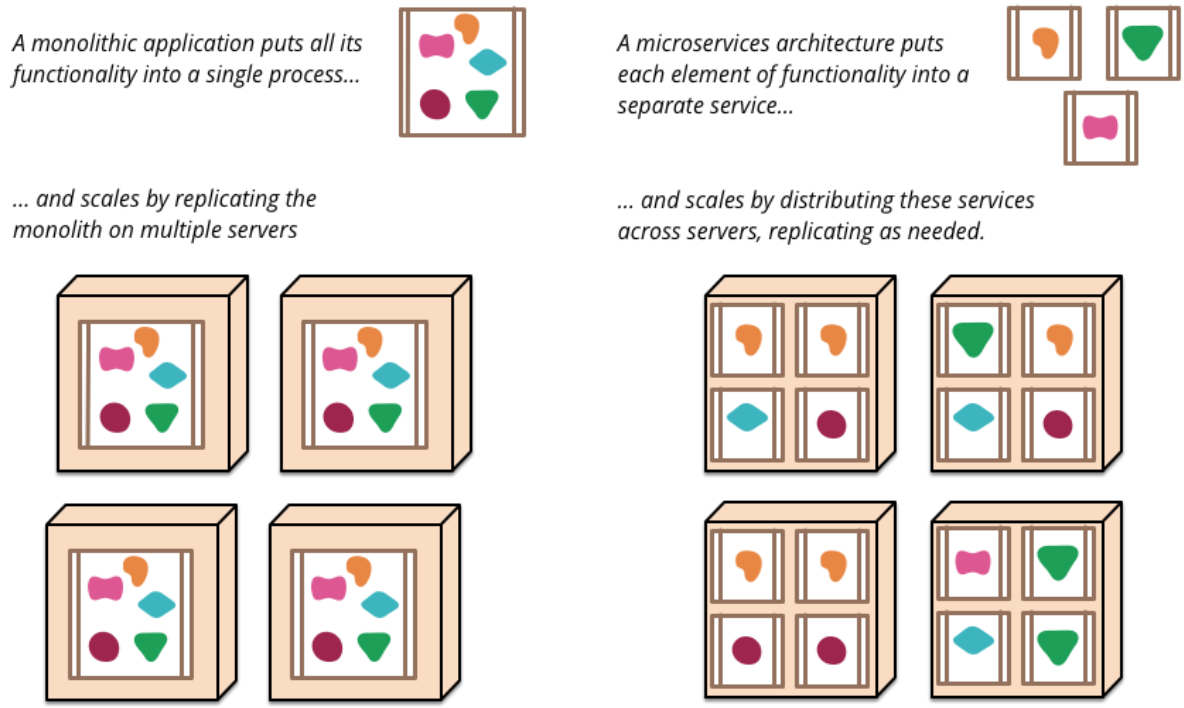


Figure 3: An example solution implemented by using pipes and filters [4]

3.2.2 MVC

In the dashboard of the system, we aim to provide many information about appliances. Thus, we will use Model-View-Controller architectural pattern which provides abstraction between data and view. The architectural design pattern will help us to isolate the domain knowledge from the user interface. In our system, model will represent the appliance data that will be analyzed and alerts that will be given by appliance. View will represent statistical representation of data. Controller will be a bridge between Model and View, analyzing the data acquired from appliances, and updating views according to this analysis.

3.2.3 Pipe and Filter

Pipe and filter design architecture consists of any number of components(filters) that transform or filter data, before passing it on via connectors(pipes) to other components.[1] The reason behind choosing this architectural pattern is convenience of task decomposition in distributed environment. This architectural pattern will be used in appliances to reduce noise in data. Additionally, the design architecture will be used in server to make analysis on data acquired from the appliances.

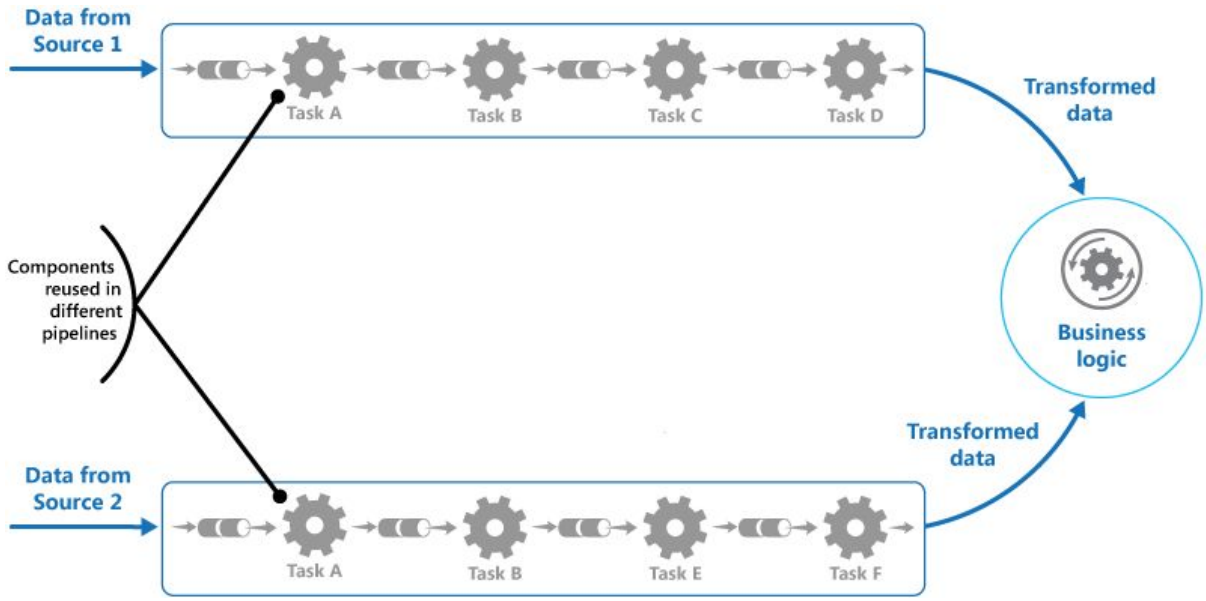
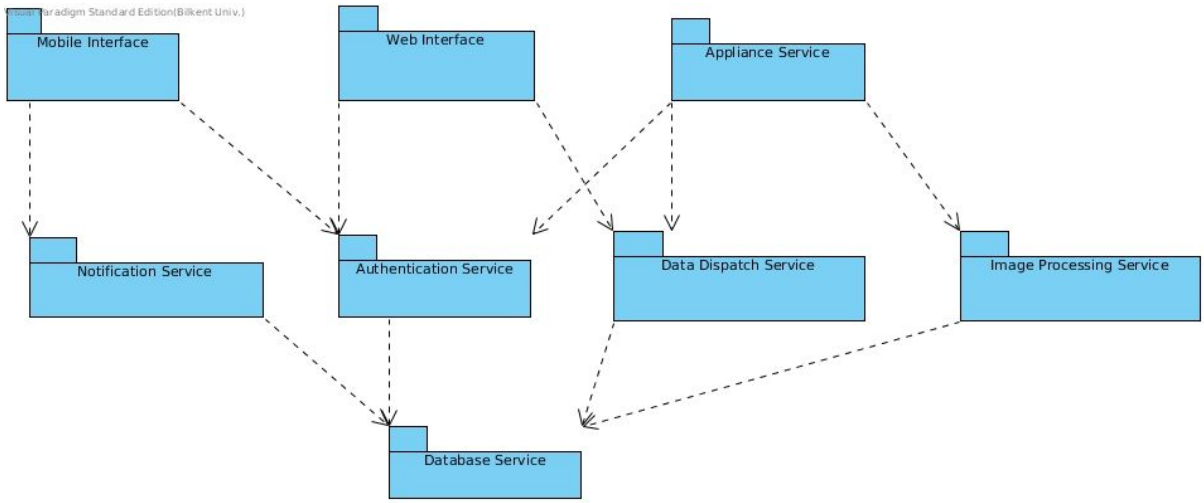


Figure 2: An example solution implemented by using pipes and filters [3]

3.2.3 Three-Tier Architecture

Horus mainframe will consist of 3 main layers which are Dashboard, Analysis layer, and Data-storage layer. Data-storage layer will consist of microservices that are related to persistent data management, Analysis layer will use the kept data and produce analysis reports that can be viewed by customers via Dashboard.

3.3 Subsystem Decomposition



3.3.1 Authentication Service

In order to register appliances to the mainframe system every appliance (after they are installed on top of legacy sensors) should be registered by using near field communication module on smart phones. Mobile application that will be on smartphones (which are previously registered to the main server) will send data such as appliance id, location data, installation date to main server. Only then, the main server should accept the sensor data sent by appliances.

3.3.2 Image Processing Service

Image data that are obtained from legacy sensors will be processed on the appliance. This method allows the appliances to send data to main server at a high frequency, since the data that is sent will not be of high volume (not raw image data).

3.3.3 Data Dispatch Service

Appliances will send processed sensor data to the main server via gsm module. Data dispatch system will prepare data to be sent, and use the gsm module.

3.3.4 Notification Service

The main purpose of this system is to track all of the user-defined filters on their appliances' data, and notify users with their own defined way. Notifying users can be done via smartphone notifications, emails, even sms texts. Content of the notification will be automatically deducted from the filter itself as a human-readable text.

3.3.5 Database Service

Database Service stores or provides the information that can be sent or requested by other services of Horus, respectively.

3.3.6 Mobile Interface

This interface is dedicated to identifying appliances and recording information in back-end services. Also mobile interface will be used to send notifications to clients in case of fired triggers.

3.3.7 Web Interface

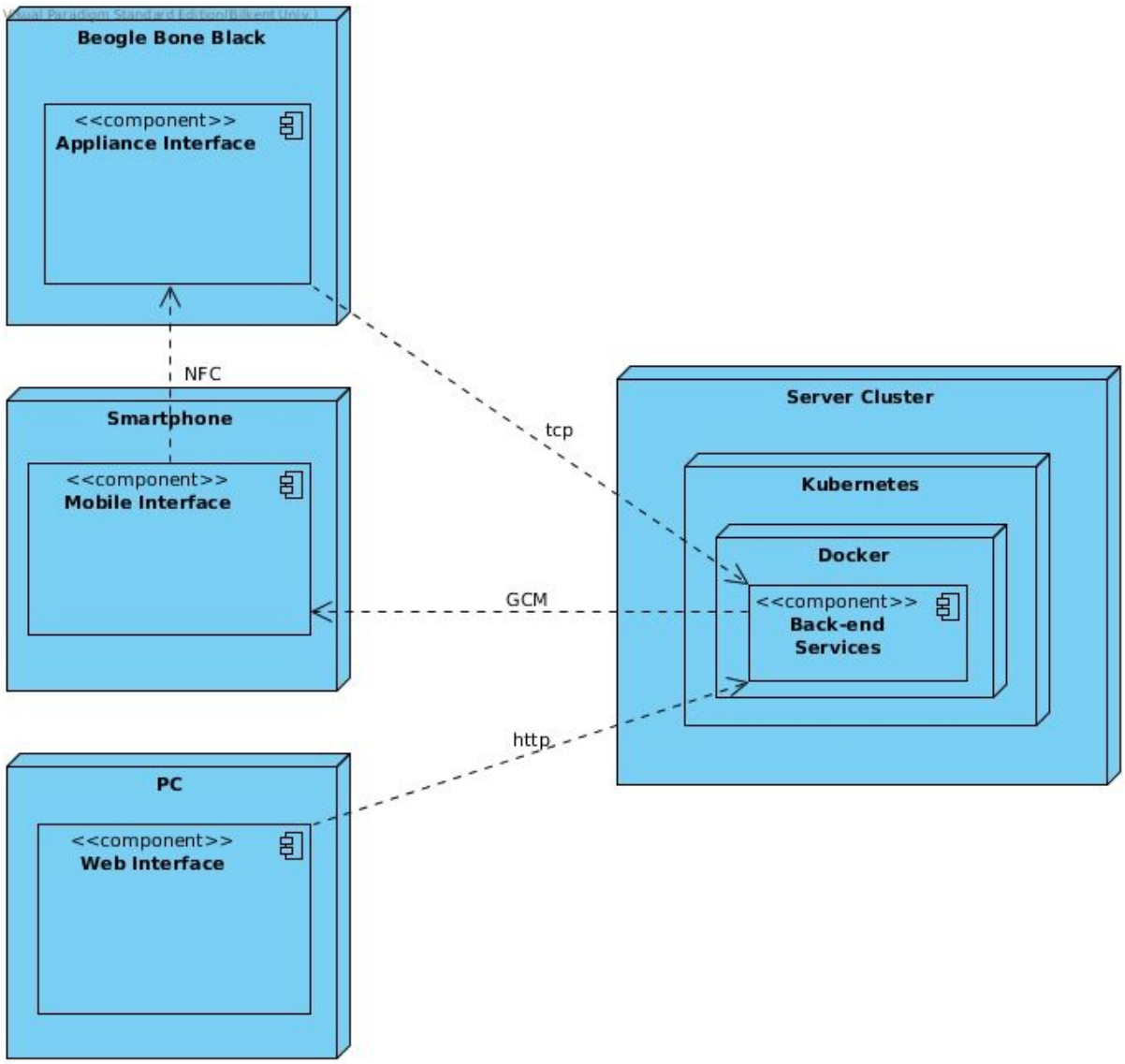
Like Mobile Interface, Web Interface service is also dedicated for controlling the interface elements for the web application of Horus and these elements are based on the
information from other services like Notification Service, Data Dispatch Service, Authentication Service, User Messaging Service. Web Interface services's main aim is to change the web interface elements according to information retrieved from other service. Web
Interface service can get information from all other services that are designed for Horus's features.

3.3.8 Appliance Service

Appliance Service, will be resided in device we will develop and it will depend on Image Processing Service. It is dedicated to extract data from camera and it will send the data it extracted using Data Dispatch service.

3.4 Hardware/Software Mapping

Hardware/Software mapping aims to map between software components and connectors of the system and the hardware parts of the application on which the software executes. In other words, it describes how subsystems are assigned to hardware and off-the-shelf components. It also lists the issues introduced by multiple nodes and software reuse. Following figure depicts the hardware/software mapping of Horus.



Horus's back-end services will reside on numerous amount of servers in order to handle some of our non-functional requirements such as fault-tolerance etc. In order to achieve these non-functional requirements we will separate servers that handle UI requests and those of which handle heavy computation. Since Image Processing Service will reside on our device, heavy computation will just consist of data processing.

In order to not depend on a specific cloud provider, we will design our system as Docker components and will use technologies like Kubernetes and Docker Fleet to scale things

up. By doing that, we will ensure that we can change our cloud provider without any hassle and we will be in a position to change them based on just economical reasons.

Our appliance will be based on BeagleBone Black, since it is a specialized circuit that is designed for on-board image processing tasks. We will be using off-the-shelf components to modify BeagleBone Black in order to meet our specific requirements.

3.4 Persistent Data Management

The sensor data collected from devices, the user information (login, device, dashboard and access information) will be the persistent data of Horus. These data will be stored securely using a database with redundant copies in order to improve data reliability.

Since Horus is a data-heavy, multiple-user application, database concurrency will become important, thus the transactions of each user and device will be logged in order to increase fault tolerance, reversibility, and recoverability in case of data failures.

3.5 Access Control and Security

Horus is a multi-user system and also each user have different kinds of access levels. The users of Horus are, admins, client admins and restricted clients. Each users have different levels of access, and have different tasks that can be accomplished.

Admins can add client admins to Horus system, and authenticate the validity of client admins. Client admins have write and read access to the system, they can edit the dashboards for the user for their devices, they can manage the devices that are tied with

their accounts, they can confirm the devices added by restricted clients and finally they can manage the restricted client accounts that are tied with the main account. Restricted clients only have read-only access to the main accounts that they are tied to, thus, they can only view the dashboards and device details and also they can add devices to the account with the "**Add Device**" scenario, that will be confirmed by their client admins.

In order to implement different types of users, user roles will be used and an access matrix like the following will be implemented.

	Add Clients	Add Restricted Clients	Manage Client Devices	Add Client Device	Confirm Client Device	Manage Dashboards	View Dashboards	View Client Device Details
System Admin	+	-	-	-	-	-	-	-
Client Admin	-	+	+	+	+	+	+	+
Restricted Client	-	-	-	+	-	-	+	+

3.6 Global Software Control

Horus is an event and data driven application. There is a constant data flow between appliances and server, and the Horus back-end performs necessary analysis, takes necessary actions solely based this data.

The events are the user-defined alert conditions defined on their sensor data transmitted by appliances. When an alert is triggered, the actions that are also defined by users on each alarm start to execute. For instance, these actions can be SMS notifications and email notifications. These events are detected by “Notification” subsystem in the back-end, and the necessary actions are executed accordingly. Also, the effects of these events are also delivered to the front-end and necessary changes will be made on the front-end views.

Other than the events, the data itself also dynamically updates the dashboard of the user with a granularity adjusted by user based on the requirements of the application.

3.7 Boundary Conditions

- **Initialization:** The system will be initialized by starting the database server and deploying the web-application to the web-application server.

- **Termination:** The system will only be terminated in case of a software update or maintenance. In this case, first the web-application server will be taken offline for a maintenance or update then if necessary the database server will be stopped.

- **Exception handling**:
    - Exception handling in distributed systems is a very challenging problem. Because of this, we are dividing exceptions into categories. Connection failure between distributed system components:
    - Appliance failures: The appliances should provide continuous data flow to the servers and there might be various physical and logical failures that might occur that hinder such data flow. For example, the image process service of the appliances might stop working accurately due to weather conditions (icing etc.), and physical conditions (dirt on sensors). In this case the image processing algorithm should notice such inaccuracies and inform the users by generating alerts that inform the appliance owners that eliminate these physical hindrances. Other than that, the some appliances might operate on battery and their operation may fail in case of low battery power. In such cases, the appliances should also generate alerts to inform the users about low battery power and request a replenishment.
    - Connection failure between appliances and servers: The connection of appliances to the servers are performed using GSM and GSM connection may fail due to various reasons. In such cases, first the appliances try to reconnect GSM multiple times and if they cannot reconnect, as the servers expect continuous data flow from appliances, they need to notice such disruptions of data flow, and generate alerts to notify users about the connection failures.
    - Network Partition and Loss of State: In any case of network partition, we will gracefully fail our systems and because of the strong decoupling of our components, we will make the system run in a restricted fashion.

4.0 References

[1] http://www.dossier-andreas.net/software_architecture/pipe_and_filter.html

[2] https://smartbear.com/learn/api-design/what-are-microservices/

[3] https://msdn.microsoft.com/en-us/library/dn568100.aspx

[4] http://martinfowler.com/articles/microservices.html