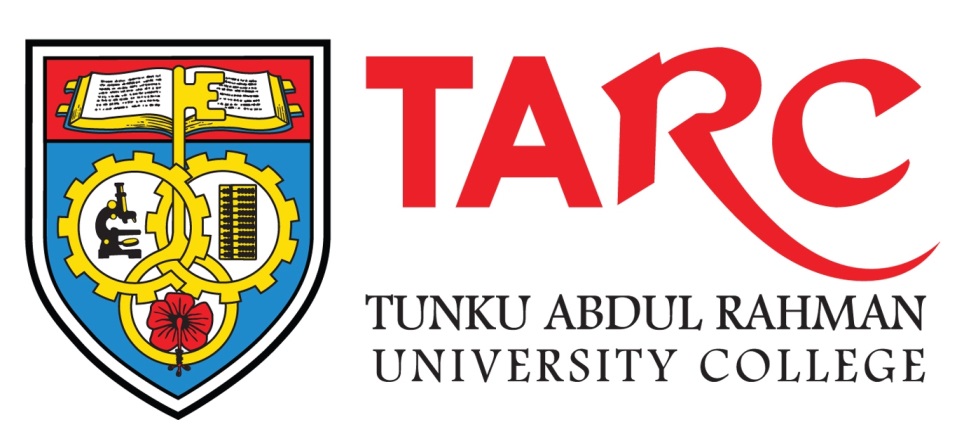
PUBLISH AND SUBSCRIBE MODULE: DYNAMIC EVENT SUBSCRIPTION

By

WANG I - SHENG



FACULTY OF APPLIED SCIENCES AND COMPUTING

TUNKU ABDUL RAHMAN UNIVERSITY COLLEGE

KUALA LUMPUR

ACADEMIC YEAR

2014/2015

PUBLISH AND SUBSCRIBE MODULE: DYNAMIC EVENT SUBSCRIPTION

By

WANG I - SHENG

Supervisor: Miss LIM YEE MEI

A project report submitted to the

Faculty of Applied Sciences and Computing

in partial fulfillment of the requirement for the

Bachelor of Computer Science

And

Bachelor of Science, Campbell University, U.S.A,

**Department of Computer Science and Mathematics**

Faculty of Applied Sciences and Computing

Tunku Abdul Rahman University College

Kuala Lumpur

2014/2015

Copyright by Tunku Abdul Rahman University College.

All rights reserved. No part of this project documentation may be reproduced, stored in retrieval system, or transmitted in any form or by any means without prior permission of Tunku Abdul Rahman University College.

# Declaration

The project submitted herewith is a result of my own efforts in totality and in every aspect of the project works. All information that has been obtained from other sources had been fully acknowledged. I understand that any plagiarism, cheating or collusion or any sorts constitutes a breach of College rules and regulations and would be subjected to disciplinary actions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Wang I - Sheng

Bachelor of Computer Science (Honours) in Software Engineering

# Abstract

In the publish/subscribe paradigm nowadays, the information providers disseminate publications to those consumers who have expressed interest by registering subscriptions. Matching efficiency and expressiveness are two key design goals of publish/subscribe systems. In this project, we have expanded the publish/subscribe system in various algorithms that send events to the owners of subscriptions satisfied by those events. The purpose of this project is that it must be able to communicate with third party systems, the event generator and the subscriber, in a modular manner. Central to the module will be a database table that records the events. Associated with the event will be a list of subscribers interested in the event. The challenging part of our scope is the event generating system comes in various forms that have to be provided data via socket, XML file, embedded API call and database tables. The methodologies that we have used mainly are research from internet and did literature review about the topic that related to research questions and come out a framework and prototyping. However, our level of achievement for this project is uncompleted. The reason is because the weakness of our project was lack of resources and tool supported. Oppositely our strength of our developed system is successfully informed the subscribers via a cell phone through SMS and sent to another system via socket in an efficient way. On the whole, all existing publish/subscribe systems cannot capture uncertainty inherent to the information in either subscriptions or publications. Therefore, the main contribution of our work is that it greatly improves the expressiveness of the publish/subscribe system without the sacrifice of matching efficiency. Last but not least, our suggestion is the module has to be able to cater for future needs without the need to change a lot of code.

# Acknowledgement

First and foremost, I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our final year project supervisor, Miss Lim Yee Mei, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

Besides that, I would like to acknowledge with much appreciation to the company HT foundation provided this opportunity in order to let final year students to accomplish this project. I am grateful to our director of HT foundation, Mr Jimmy Ng for his continuous support for the project, from initial advice and contact in the early stages of conceptual inception and through ongoing advice and encouragement to this day.

A special thank of mine goes to my team mate, Tan Ze Ming who helped me in completing the project and his exchanged his interesting ideas, thought and made this project easy and accurate.

Furthermore, I wish to thank my parents for their undivided support and interest who inspired me and encouraged me to go my own way, without whom I would be unable to complete my project.

At last but not the least I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved out presentation skills thanks to their comment and advices.

# Table of Contents

[Declaration iii](#_Toc381544644)

[Abstract iv](#_Toc381544645)

[Acknowledgement](#_Toc381544646) v

[Table of Contents vi](#_Toc381544647)-vii

[1. Introduction 1](#_Toc381544648)

[1.1 Project aims and objectives 2](#_Toc381544649)-3

[1.2 Company background 3](#_Toc381544651)

[1.3 Project Scope 4](#_Toc381544652)

[1.4 Project Schedule 5](#_Toc381544652)

[1.5 Methodology used 5](#_Toc381544652)

[1.6 Development and Operational Environment 5](#_Toc381544652)

[1.7 Chapter Summary and Evaluation 6](#_Toc381544652)

[2. Literature Review 7](#_Toc381544653)-9

[3. Methodology 11](#_Toc381544656)

[3.1 Software Development Model 12](#_Toc381544657)-13

[3.2 Fact Gathering Techniques 13-14](#_Toc381544658)

[3.3 Fact Recording Techniques 15](#_Toc381544659)

[3.4 Chapter Summary and Evaluation 16](#_Toc381544660)

[4. Requirements Analysis 17](#_Toc381544661)

[4.1 Functional Requirements](#_Toc381544662) 18-20

[4.2 Main Function – Capture events](#_Toc381544662) 20-21

[4.3 Non-functional requirements](#_Toc381544662) 21-22

[4.4 External Interface Requirements](#_Toc381544662) 22-23

[4.5 Other requirements](#_Toc381544662) 23

[4.6 Chapter Summary and Evaluation 23](#_Toc381544663)

[5. System Design 24](#_Toc381544664)

[5.1 Publish/Subscribe framework 25](#_Toc381544665)

[5.2 Context Diagram of pub/sub module 26](#_Toc381544665)

[5.3 Overall Architecture of publish/subscribe system 27](#_Toc381544665)

[5.4 Subscribe module basic research model overview 28-29](#_Toc381544665)

[5.5 Use Case diagram 30-33](#_Toc381544665)

[5.6 Database Design 34-36](#_Toc381544665)

[5.7 Class Diagram 37-38](#_Toc381544665)

[5.8 Subscribe interaction sequence diagram 39-41](#_Toc381544665)

[5.9 Algorithms research framework 42-51](#_Toc381544666)

[6. Programming 52](#_Toc381544667)

[6.1 ASP.NET framework 53](#_Toc381544668)-56

[6.1 Code analysis 57-59](#_Toc381544668)

[6.2 Chapter Summary and Evaluation 59](#_Toc381544669)

[7. Software Testing 60](#_Toc381544670)

[7.1 Software Testing Methods 61](#_Toc381544671)

[7.2 Results 62-65](#_Toc381544672)

[7.3 Chapter Summary and Evaluation 65](#_Toc381544673)

[8. Conclusion 66](#_Toc381544680)

[9.1 Summary 67](#_Toc381544681)

[9.2 Completeness of the Project 67](#_Toc381544682)

[9.3 Contributions 68](#_Toc381544683)

[9.4 Future Improvements 68](#_Toc381544684)

[9.5 Issues and Solutions 68-69](#_Toc381544685)

[9. References/Bibliography 70](#_Toc381544686)-71

[10. User Guide 72](#_Toc381544687)

[11.1 System Document 72-75](#_Toc381544688)

[11.2 Operation Document 75-80](#_Toc381544689)

[11. Appendices 81](#_Toc381544690)

[A.1 Other Existing Algorithms 81](#_Toc381544688)

[A.2 Data Structures and Algorithms 81-82](#_Toc381544688)

[A.3 Corona 82-83](#_Toc381544688)

[A.4 Publish/Subscribe with database integration 83](#_Toc381544688)

[B. HT Foundation Company’s Requirements 84-88](#_Toc381544688)

[C. Experiment Test Plan and Result 89-90](#_Toc381544688)

Chapter 1

Introduction

# Introduction

In the publish/subscribe paradigm, information providers disseminate publications to all consumers who have expressed interest by registering subscriptions. This paradigm has found wide-spread applications, ranging from selective information dissemination to network management. Recently, there has been a lot of research on solving the problem of efficiently matching events against subscriptions. The main existing problem is there are many type of event in publish/subscribe, but no one system able to capture all type of event currently. Besides that, there are different types of subscribers and each of them may use different channel to subscribe their interest information. The channel can be mobile, third party system, television, e-mail, etc. Unfortunately for this moment, none of such systems are able to disseminate the information to all kind of channel. However, existing matching algorithms are limited.

The reason we want to solve this existing problems is it should have strong expressiveness. For instance, it should support events in different formats and semantics. Furthermore, it should provide a powerful subscription language, so that information consumers can easily express their interest in certain events. Moreover, especially in selective information dissemination applications, it is often more appropriate for a user to formulate his/her search requests or information offers in less precise terms, rather than defining a sharp limit.

The remainder of the paper is organized as follow. In chapter 2, the literature review that we have done. In chapter 3, we discuss the methodology that we been used. In Chapter 4, we have provided the detailed results from the fact gathering. In chapter 5, we have the various areas of design for the new proposed system. In chapter 6, we have included a description of the various tools and techniques used in both the program designs and program development. In chapter 7, we have produced an experiment plan that utilizes various software testing strategies and methods. Finally in chapter 8, we have concluded the overall project.

* 1. **Project aims and objectives**

Although we are taking the research-based for our project, yet we have also taken the industrial-based project from HT Foundation at the same time. Consequently, we have planned to do some research in order to fulfill the requirement from both institutions. Nonetheless, the purpose in this project is to make the proposed software system become more generic in spite of we will allow this module able to capture all type of events and disseminate those events to all kind of subscriber in different channel. In the meantime, the sub purpose is to make the algorithm become more efficiently and effectively.

However our objectives for doing this project as shown at the following:

1. To analyse the various algorithms and choose the most suitable one to support various channel.
2. To improves the expressiveness of the publish/subscribe system and at the same time keeps a high matching efficiency.
3. To evaluate the performance of our algorithm and compare the effect of our clustering strategies.
   1. **Company background**

HT Consulting is an indigenous ICT Consulting, Systems Development and Outsourcing Services company founded by its visionary CEO Harres Tan in 1989. The company, which is part of the HT Group includes HT softLab which is the exclusive R&D arm of the group and Rototype International which focuses on specialized self-service solution for the banking industry. Over the past 21 years, the group has successfully provided solutions and services to industry sector ranging from Government, Healthcare, Manufacturing, and Banking and Finance. Figure 1.1 had shown the HT Foundation Company’s logo.



Figure 1.1: HT Foundation Company’s logo

* 1. **Project Scope**

Publish and subscribe module

Publishmodule

Subscribemodule

*Analyse information*

*Register an account*

*Match information*

*Capture publish events*

*Publish information*

*Update events*

Figure 1.2: Hierarchical Chart showing project sub-systems

In this project, my main task is to accomplish subscribe module and following are the sub-modules’ descriptions:

1. Capture events function is a function that captures the publication from publisher and add the subscriptions that subscriber subscribed. It will then store the publication/subscription details into database.
2. Update events function is a function that allows the subscribers to renew their subscription.
3. Register an account function is a function that allows interested users to register an account in order to and subscribe interested topics.
   1. **Project Schedule**

|  |  |  |  |
| --- | --- | --- | --- |
| No | **Task** | **Date** | **Remark** |
| 1. | Project plan and project proposal | 18/07/2014 | - |
| 2. | Literature review and requirements gathering | 19/07/2014 - 28/7/2014 | Proposal approved |
| 3. | Project specification | 29/07/2014 - 08/8/2014 | Accomplished requirement gathered |
| 4. | Research plan analysis | 09/08/2014 - 30/9/2014 | Accomplished project specification |
| 5. | Requirement analysis | 01/10/2014 - 24/11/2014 | Accomplished research plan analysis |
| 6. | Research design specification | 25/11/2014 - 13/12/2014 | Accomplished requirement analysis |
| 7. | Submission of software requirement specification | 14/12/2014 | Accomplished research design specification |
| 8. | Setup programming tools and coding implementation | 15/12/2014 – 03/02//2015 | Learn new software development |
| 9. | Generate experiment plan | 04/02/2015 - 05/02/2015 | Accomplished system implementation |
| 10. | Final testing with Supervisor and Moderator | 06/02/2015 | Accomplished experiment plan |
| 11. | Prepare draft report and final report | 07/02/2015 – 19/05/2015 | Final testing tested. |
| 13. | Adjust final report | 20/05/2015 – 09/06/2015 | Draft report submitted |
| 14. | Submission of Final Report | 10/06/2015 | - |

* 1. **Methodology used**

Due to we have chosen the research-based for this project, consequently we are mainly using literature review to accomplish our proposed paper and project. Nonetheless, we have taken industrial project from HT Foundation Company at the same time, so we have implemented the evolutionary prototyping during our system implementation part. The reason we have used this approach was because we can build a very robust prototype in a structured manner and constantly refine it.

* 1. **Development and Operational Environment**

We have used the ASP.NET MVC 5 to develop our prototype for our proposed systems. One of main reasons is that this language provides clean event management by using “delegates”. Delegates are the events are fundamental to any Windows or Web Application, allowing the developer to “subscriber” to particular actions carried out by the user (Wikibooks, 2011). Moreover, this language is more suitable for event handling as well. In a nutshell, we cannot design our framework and architecture of publish and subscribe module yet due to our resources are limited at this moment.

## Chapter Summary and Evaluation

At the earlier stage, we have made such difficult decision that whether want to choose fully research-based paper or package-based project for our project. At last, we have chosen both project and paper to be accomplished together, which are partially research-based paper mixed with industrial project. It is because we found that it was very hard to find a suitable and agreeable paper to finish up our paper due to we have to deal with Artificial Intelligence (AI)’s element in order to fulfill our final year project’s requirement. In the meantime, it was fortunate that TAR UC had dealt with industrial based from various companies, yet we have taken that opportunity to look after those companies that provided the suitable project title for students. After all, we have chosen publish and subscribe module that generated by HT Foundation Company as our project.

Furthermore, we have discussed with our TAR UC’s supervisor that whether this project title is suitable for us in order that we can accomplish our research-based paper while accomplishing the project assigned by HT Foundation. Definitely my team mate and I have done a lot of research regarding the proposal title before we discuss with supervisor. As a result, we have got an approval from both supervisor and moderator that we are able to accomplish our research-based paper while doing the industrial project.

Chapter 2

Literature Review

# Literature Review

Basically, publish – subscribe paradigm consists of three main components. There are publishers, who generate and feed the content into the system, subscribers, who specify content of their interest, and an infrastructure for matching subscriber interests with published content and delivering matched content to the subscribers. Generally, publish/subscribe systems are divided into two main categories, which are subject-based and content-based. In subject-based systems, publishers and subscribers are connected together by pre-defined topic, so called channels, each event belongs to one of the a fixed set of subjects, it may also called subjects or groups; content is published on well-advertised channels to which users subscribe to receive asynchronous updates and subscribe to all event under a particular subject. Nonetheless, publishers are required to label each event with a subject name. On the contrary, content-based systems enable subscribers to express elaborate queries on the content and use complicated content filtering techniques to match subscriber interests with published content. All events that meet the constraints of the subscription will be sent to the subscriber. However, the content based systems are more flexible and expensive than the subject-based publish/subscribe systems due to they enable subscribers to express their interests in a better level of granularity.

Besides that, the existing content-based publish/subscribe systems can be further divided into two sub-categories, which are map-based and XML based. Within the map-based systems, each event is a set of “attribute = value” pairs, and subscriptions are usually conjunctions of simple predicates on data attribute, which are so called flat patterns. While in XML-based publish and subscribe systems, each event is an XML document, and subscriptions are usually XPath expression or its variations, which contain not only constraints on the structure of the XML documents but also constraints on certain elements and attributes. Such subscriptions are called tree patterns (Milenko, 2003).

The reason we encourage to use content-based systems to filtering is the messages are only delivered to a subscriber if the attributes or content of those messages match constraints defined by the subscriber. The subscriber is the one who responsible for classifying the messages. Besides that, Benjamin et al. (2001) claimed that the advantage of a content-based system is flexibility. It can provide the subscriber just the information he/she needs and the subscriber need not have to learn a set of topic names their content before subscribing. On the contrary, Benjamin et al. (2001) also claimed that the disadvantage of content-based systems is the burden it places on the underlying system to match messages to the subscription. The number of unique subscriptions can be orders of magnitude larger than the number of groups that must be managed in the subject-based system. Hence, our matching must be done efficiently. One way to solve this problem is by using ontologies. Semantic information systems use ontology to represent domain-specific knowledge and allow users to use the ontology terms to construct queries. Besides that, this allows uniform access to multiple heterogeneous information sources. The problem of adding semantic capability to publish/subscribe systems can be seen as an “inverse” problem to the heterogeneous database integration problem. In semantic publish and subscribe events correspond to data, so now the problem is how to match data to queries (Christine, Michael, and Sheri, 2001).

Y. Arens and C.A (1992) maintain that some systems use inference engines to discover semantic relationships between data from ontology representations. Are. Inference engines usually have specialized languages for expressing queries different from the language used to retrieve data, so that the user queries have to be either expressed in or translated into the language of the inference engine. Additionally, there are systems that use mapping function exclusively and do not have inference engines. In those systems, mapping functions serve the role of an inference engine (Edward, et.al, 2004). Apart from that, web service discovery is a process of matching user needs to provide services that user needs are analogous to events and provided services to subscriptions in publish/subscribe system. Massimo et al. (2012) claim that during a discovery process, the web service discovery systems are functionally similar to a publish/subscribe system that a web service advertises its capabilities in terms of its inputs and outputs. On the whole, a user looks for a particular web service by searching for appropriate inputs and outputs according to the user’s needs. Relevant services are determined by either exact match of inputs and outputs, or a compatible match according to ontology relationship.

At the end, the above networks efficiently implement a simple form of publish/subscribe. However, the topic based approach suffers from limited expressiveness and selectivity by only allowing predefined topics. Therefore, they leave a large part of the filtering to the leaf nodes of the event dissemination graph. For content based filtering, on the other hand, aims to deliver only useful matches.

Chapter 3

Methodology

# Methodology

The two main algorithms below had dedicated what we have done for this paper and project:

1. Data-structures algorithm

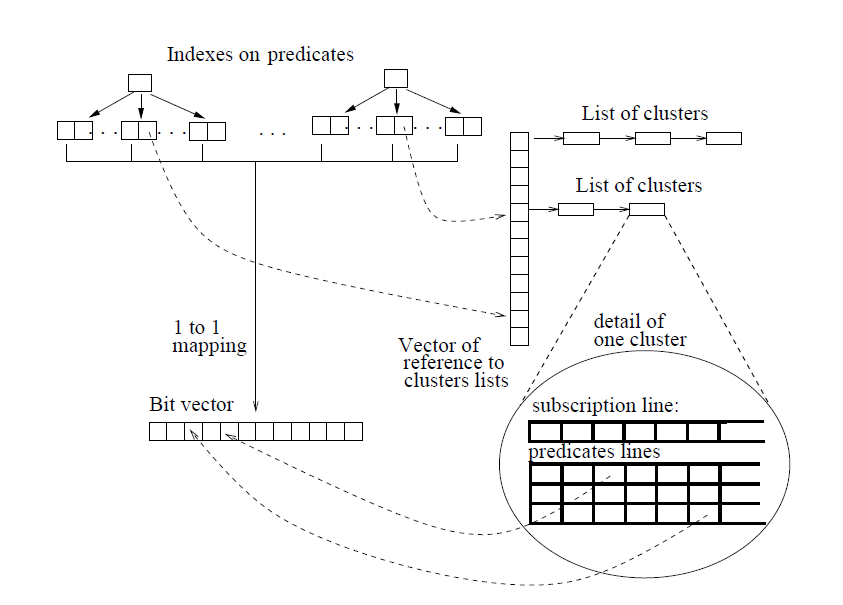
This algorithm uses a set of indexes, a predicate bit vector and a vector of references to subscription cluster lists, called a cluster vector as shown in figure 3.1. Besides that, this algorithm satisfied by a given incoming event that uses the indexes to compute the set of predicate, and the set of clusters which are related for the event.

Figure 3.1: Data-structure algorithm (Fabret, F, 2001).

1. Event Matching algorithm

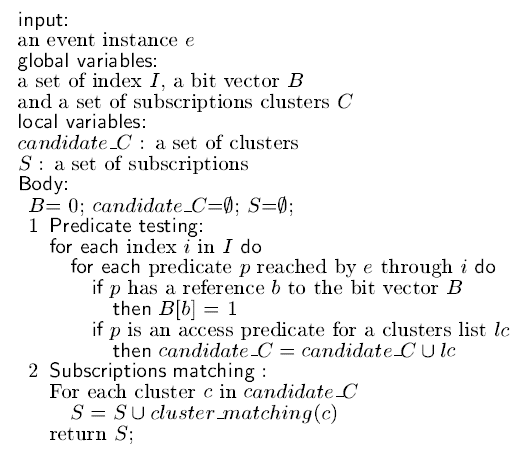
This algorithm is executed each time when the new events come in and it had consists of two steps as shown in figure 3.2.

Figure 3.2: Event matching algorithm (Fabret, F, 2001).

1st step: Uses the indexes to compute the set of verified predicates, it set to 1 all corresponding entries in the predicate bit vector and collects the lists of clusters having verified access predicates.

2nd step: Considers each candidate cluster and applies the “*cluster\_matching”* algorithm to compute matching subscriptions.

## Software Development Model

Besides using the method of literature review to accomplish our project, we have also implemented the evolutionary prototyping. It is a prototype model and the main goal is to build a very robust prototype in a structured manner and constantly refine it. The reason that we have used this model is once the evolutionary prototype had built, forms the heart of the new system and the improvements and further requirements will be built. While the developers doing their project using the evolutionary prototyping, they can focus themselves to develop parts of the system that they understand instead of working on developing a whole system. In order to minimize risk, the developer does not implement poorly understood features and the partial system is sent to customer sites.

However, our main advantage of using this model is it can improve and increase user involvement. It requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specification. Since users know the problem domain better than anyone on the development team does, increased interaction can result in a final product that has greater tangible and intangible quality. On the contrary, the disadvantage is the excessive development time of the prototype. A key property is the fact that it is supposed to be done quickly. Once the developers lose sight if this fact, they may try to develop a prototype that is too complex. Moreover, users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product. On the whole, our project will be slightly trend to disadvantage due to our developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype.

* 1. **Fact Gathering**

After we have reviewed many papers, direct notification of subscribers by a publisher is not scalable. Therefore, it is vital to develop techniques for distributing the notification process. Nevertheless, a number of pub/sub network designs have been developed.

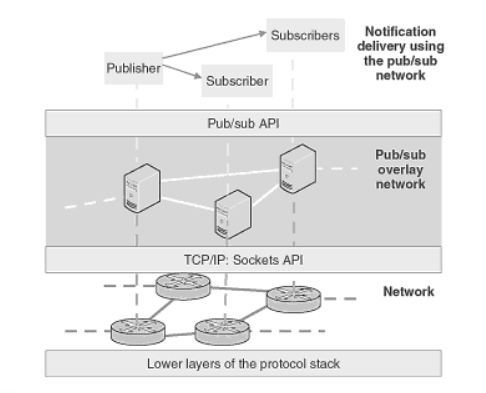
First and foremost, Tarkoma (2012) contend that an event broker or router is a component of a pub/sub network that forwards notification messages across multiple hops. An example pub/sub network is presented in Figure 3.3 that shows the layered design. The pub/sub network offers the notification API to subscribers and publishers and utilizes the network API, typically the Sockets API, to disseminate the notification message and take it from the source router to the destination router and subnet work. The network level routers are responsible for taking the message end-to-end across the Internet. Such overlay designs have favourable characteristics in terms of deplorability and flexibility. However, the resulting high level routing may not be efficient in terms of the network level topology.

Figure 3.3: Example of pub/sub overlay network

Secondly, an event router typically has local clients and neighbouring routers. The algorithms and protocols for local clients and neighbouring routers are different. Both cases require a routing table for storing information about message destinations. A pub/sub routing table is an index structure that contains the active subscriptions and typically supports add, remove, and match operations.

The design and configuration of pub/sub networks has become an active area of research and development. We will focus on various strategies for implementing pub/sub networks. The simplest form of notification in the distributed environment is called flooding. With flooding each pub/sub broker simply sends the message to all neighbours except the one that sent the message. Thus the message is introduced at every broker. However, the price of the technique is its inaccuracy. Ideally, we want to prevent the forwarding of a message to a broker that we know does not have subscribers for the message. Moreover, excess and uncontrolled messaging may lead to congestion that in turn may cause notification messages to be dropped.

* 1. **Fact Recording**

**Publish Publications**

**Subscriptions**

**Subscribe using Boolean expression**

**Publish the interest event**

**Publish and subscribe system**

**Retrieve**

**Manage Publications**

**Database A**

**Publishers**

**Send Signal**

**Subscribers**

**Publish module**

**Subscribe module**

## Chapter Summary and Evaluation

This chapter has outlined the methodological framework utilised within this study. Congruent with the grounded theory approach and constant comparative analysis have been central to this study. The ethical aspects associated with this research have been addressed within the chapter and the steps taken to ensure rigour and trustworthiness have been discussed.

Apart from that, the problem we have faced at the beginning stage is that we were not sure which development model that we going to implement. The reason is there are various type of software development processes to use in software engineering method and we don’t know which one will be the best model to implement it due to our main proposal of this project is using the research method, yet we have taken over the industrial project from HT foundation at the later time.

However, after we have consulted our TAR UC’s supervisor and the company’s moderator, my team mate and I have decided to use both methods to accomplish our project which are literature review and evolutionary prototyping.

Chapter 4

Requirements Analysis

# Requirements Analysis

After the notification has been published, it is the duty of the pub/sub system to deliver the message to interested recipients – the subscribers. A subscriber is an entity that has expressed prior interest to a set of events that meet certain requirements that the subscriber has set. The actual delivery depends on the pub/sub solution being used. For example, it could be based on the following:

1. The message is broadcast on the network and devices on the same network will see the message. The pub/sub system running on a device can then process the message and deliver it to the subscriber if it is active on the device.
2. The message is delivered via network supported multicast, in which a specific network primitive is used for delivering the message from one publisher to many subscribers.
3. The message is sent directly by the publisher to subscribers that have informed the publisher that they are interested in receiving a notification. The publisher then utilizes a one-to-one message delivery protocol on top of the communication primitives offered by the network, typically the TCP/IP protocol stack.
4. The message is first sent to a broker server and then delivered by the broker to active subscribers. In this case, the subscribers have expressed their interest in receiving notifications with the broker.
5. The message is delivered through a network of brokers. The scalability of a pub/sub system can be increased by deploying a network of pub/sub brokers.
   1. **Functional Requirements**

This section includes the requirements that specify all the fundamental actions of the software system.

* 1. Register an account (Add new subscriber)
  2. The function consist the attributes of user name, check availability, date of birth (DOB), contact number, e-mail address, password, and re-type password.
     1. The user name is a text field form and the default text field is empty.
     2. The DOB is a calendar form and default text field is empty.
     3. The contact number is a text field form and the default text field is empty.
     4. The e-mail address is a text field form and the default text field is empty.
     5. The password is a text field form and the default text field is empty.
     6. The re-type password is a text field form and the default text field is empty.
  3. System will verify all the inputs.
     1. The user name text field cannot be empty. Otherwise, “Please enter user name” error message will be prompted. Besides that, the first character cannot be space or symbol form.
     2. The DOB calendar form cannot be empty. Otherwise “Please enter DOB” error message will be prompted.
     3. The contact number cannot be empty. Otherwise, “Please enter phone no.” error message will be prompted. Besides that, the format of contact number must consist of “ – “. Otherwise, “Please enter proper format of phone no.” error message will be prompted.
     4. The e-mail address text field cannot be empty. Otherwise, “Please enter e-mail address” error message will be prompted. Besides that, the format of e-mail address must consist of “@” and “.com” within the email’s name. Otherwise, “Please enter proper format of e-mail” error message will be prompted.
     5. The password text field cannot be empty. Otherwise, “Please enter password” error message will be prompted. Besides that, password’s length must be between 4 – 16 characters and consists alphanumeric with a symbol. Otherwise, “Please enter only 4 – 16 characters” error message will be prompted.
     6. The re-type password cannot be empty. Otherwise, “Please enter re-type password” error message will be prompted. Besides that, if the re-type password does not match the text from password text field, it will prompt user the error message of “Re-type password incorrect, please enter again!”.
  4. The new button allows the user reset all the user inputs.
     1. The text field of user name will be in empty field.
     2. The calendar form of DOB will be in empty field.
     3. The text field of contact number will be in empty field.
     4. The text field of e-mail address will be in empty field.
     5. The text field of password will be in empty field.
     6. The text field of re-type password will be in empty field.
  5. Capture events (Add publications/subscriptions)
  6. The events types drop down list allows the user to choose whether want publish or subscribe events. The items are “publication” and “subscription” under the events types drop down list.
     1. The publication item will bring user to another page that can see the publication that user been published. Besides that, there is an option list to let user to upload the new event.
     2. The subscription item will bring user to another page that can see the all the existing events. User may click the “subscribe” button if he/she are interested.

1. Update existing events (Renew subscriptions)
   1. The “Renew” button allows the admin to update the corresponding user subscriptions.

3.1.1 The system will prompt user to select whether want renew directly “1 year” or “2 year” at once.

* 1. The system will verify the subscriptions of subscriber after subscriber clicked “Renew” button.
  2. System will prompt user a dialog box with “Successfully Renew” after user clicked the “Renew” button.
  3. System will update the subscription into the database.
  4. **Main Function – Capture events**

Inputs are created in a diverse range of formats, may comprise multiple individual objects (compound inputs), and are transmitted by a wide range of communication channels such as workflows, e-mail, postal mail, etc. Pub/sub system must capture the content, structure and context of events to ensure they are reliable and authentic representations of the business activities or transactions in which they were created or transmitted. This is known as “point of capture” metadata and should in itself be captured as an event. Besides that, it should not be possible to alter any of these metadata features without changes being tracked and auditable.

The following had dedicated that the minimum requirement that main function have:

1. Enable integration with business applications so that transactional inputs created by those applications can be captured within the pub/sub system.
2. Indicate when an individual input is captured within the pub/sub systems.
3. Prevent the alteration of the content of any input by any user or administrator during the process of events capture.
4. Alert a user to any failure to successfully capture an event.
5. Ensure that all records captured within the pub/sub system are associated with at least one aggregation.
   1. **Non-functional requirements**

*4.3.1 Availability*

Many services are subject to a variety of availability attacks. Replay is a common attack and it is recommended that this be addressed as described in the next bullet. Other attacks, such as network-level denial of service attacks are harder to avoid and are outside the scope of this specification. It could be said that care should be taken to ensure that minimal processing be performed prior to any authenticating sequences.

*4.3.2 Reusability*

We have identified two risks related to reuse for invocation: limitations on system structure due to deadlock, and potential for architectural mismatch. We have already observed that deadlock should not be an integration problem with notification. There is also less risk of architectural mismatch because the data source always initiates the interaction. Of course, components must agree on data type to exchange data in any case. Evocation could also have a mismatch if the interface-method signatures were not the same. However, publish/subscribe does not suffer from this limitation.

*4.3.3 Security*

1. Only authorized principals can subscribe for notifications.
2. Only authorized principals can modify or delete subscriptions.
3. Only authorized principals can receive the notification message.
4. The system shall transfer data securely via encryption.

*4.3.4 Reliability and Predictability*

Mitchell et al. (2007) claim that when considering invocation and notification, we thus know that there is an obvious change between reliability and predictability. Using invocation, the calling process waits for and receives a result, guaranteeing that the remote method has completed successfully. The change for this reliability is the fact that the time required to complete an invocation depends on the object and method invoked, the current load of the processor it resides on, and the current network load.

With notification, the sender must assume that the message will be received and processed correctly. The time required to complete a notification is only the time required to complete a notification is only the time to create and send a message, which is independent of the object(s) being notified. Obviously, notification would be slightly less predictable if distribution semantics were not optimized with some multicast or broadcast messaging scheme since it would create and send a message to each receiver.

*4.3.5 Performance*

1. Response time for a transaction (average, maximum).
2. The amount of material or items passing through a system or process. For example, transactions per second.
3. Capacity. For example, the number of customers or transactions the system can accommodate.

**4.4 External Interface Requirements**

*4.4.1 User interfaces*

For user interface design, common GUI standards will be followed along with the presence of keyboard shortcuts, error message display standards etc. and standard buttons and functions will appear on every screen. This section describes the logical characteristics of each interface between the intended software product and the users. Details of the user interface design intended to be documented in a separate user interface specification.

*4.4.2 Hardware Interfaces*

It is intended that the pub/sub system will be deployed within the context of the certain project. Therefore, the Fleck sensor of hardware interfaces will be used to program the deployed sensor nodes. In addition, common hardware interfaces for gateway operation and communication will be used.

*4.4.3 Software Interfaces*

There will be a Web-based interface for the intended product, through which client applications can subscribe to the published service configurations and get notified. Relevant Tomcat server and MongoDB database, ASP.NET libraries will be used to make a complete concept demonstrator of the product. Details of the software interface will be included as the product goes through full-fledged development.

*4.4.4 Communication Interfaces*

Various application program interface (API) will be used to notify subscribing client applications. The rest of the Web services APIs will be exposed to communicate with the product and integrate into a deployed sensor network. Details will be available over the development of the product.

* 1. **Other requirements**
     1. *Performance requirements*

The performance of the system will be tested under different scenarios and results will be demonstrated in terms of performance metrics such as response time, scalability, throughput etc. Detailed performance evaluation will be carried out to demonstrate superior performance of the product.

* + 1. *Software Quality Attributes*

It is important to the developers or customers about the additional quality characteristics of the product. Those additional quality characteristics included adaptability, availability, correctness, flexibility, maintainability, reliability, testability, usability, robustness, interoperability, portability and reusability. These additional attributes will be elaborated as the product reaches maturity.

* 1. **Chapter Summary and Evaluation**

This chapter has outlined the requirement analysis within this study. The main issue that we had faced were functional requirements. Due to we have lack of this kind of knowledge of the propose system, we only can list out limited functional requirement of our system. Based on the normal user or novice user, they may only know how to subscribe certain events in order to up-to-date themselves to get the latest information. But for us, besides those common features, we have to analyse what else the functions have in the corresponding module. However, we have did a lot of research from internet and reference books and sought for assistance from our company’s supervisor, consequently we have gained quite a lot of ideas that what we are going to do have in the system.

Besides that, the fact analysis is most difficult section that we need to be done. Due to we are doing research based for our final year project, so we can’t provide any statistical method in this software requirement specification. What we replaced in this section was the result of experiment plan.

Chapter 5

System Design

* 1. **Publish/Subscribe framework**

Partner Service Component (Devices/Laptops/PCs/Server))s)

Notification Service Provider (With policy engine)

Topic Registry

Web Service

Message Handler

Web Service Client

Publisher Handler

Outbox

Policy Database

Policy Engine

Web Service Client

Web Service

Subscriber Handler

Inbox

Figure 5.1: Policy-based publish/subscribe framework (Benjamin et al., 2010)

Figure 5.1 had shown that Partner Service Component can publish a continuous stream of events associated with a participant to the notification service provider. In the meanwhile, it enables participate to subscribe to a continuous stream of events, which is generated by other participants that are delivered to the Partner Service Component by the Message Broker. Those events are transmitted as messages, with published messages being sent from an Outbox managed by the Partner Service Component, and subscribed messages being delivered to an Inbox (Peyton et al., 2007).

Secondly, the notification service provider maintains a registry of the types of events, called Event Topics, and the types of data, called Event Attributes, used to define those topics. It is to ensure a consistent event data model across the network. Besides that, Huhns and Singh (2005, p.75) acknowledge that message broker is also the central policy enforcement for the network since all sharing of event data through Partner Service Component subscriptions that must pass through it.

Lastly, the publish/subscribe framework allows for dynamic routing of message. In a traditional Service-Oriented Architecture (SOA) approach subscribers would either have to poll and continually check if there were messages for them, or a predefined process would deliver message to predefined recipients (Eidson et al., 2005).

* 1. **Context Diagram of pub/sub module**

**Publish Publications**

**Subscriptions**

**Subscribe using Boolean expression**

**Publish the interest event**

**Publish and subscribe system**

**Retrieve**

**Manage Publications**

**Database A**

**Publishers**

**Send Signal**

**Subscribers**

**Publish module**

**Subscribe module**

* 1. **Overall Architecture of publish/subscribe system**

Consumers (subscriber application)

Producers (publisher applications)

Event 2

Event 1

Thrift based API

Management & Monitoring

Event 3

Broker

The main components of our centralized database are:

1. *Persistent Queue*- Physical implementation of logic topic, internally use memory mapped file, automatic paging and swapping algorithm, sliding window, index based access for fast queue operation while use memory in an efficient way.
2. *Thrift based interface*- Simple based API exposing queue service to external clients
3. *Management and Monitoring*- Server management and Java Management Extensions (JMX) based monitoring interface.
   1. **Subscribe module basic research model overview**



Notification Service Provider

Publish

Capture()

Centralized database:

Storage of publication and subscription

Unsubscribe()

Subscribe()

Unsubscribe

Subscribe



**Evaluation of subscribe model**

Information has been referred to as Notifications in this paradigm and it described as events or publications produced by publishers that subscribers interested in and notified when they are available. Subscribers can continue their tasks until the notification service delivers notifications. The advantages of subscribe model as shown below:

* Enhanced response time
* Enhanced results
* Database resources utilization and increased capacity
* Loosely coupled relationship between publishers and subscribers
* Scalability

Decoupling types between publishers and subscribers have involved space decoupling in which publishers and subscribers do not need to know each other. Besides that, it may involve time coupling in which publishers and subscribers do not need to be running at the same time. Additionally, it may also involve synchronization decoupling in which publishers and subscribers operations and task are not halted during publishing and receiving notifications. However, scalable system that fits well in distributed systems.

Other communication models existed aside from publish-subscribe model:

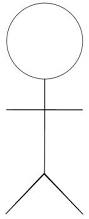
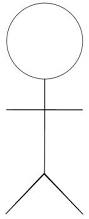
1. *Message passing*:

* The message production done asynchronously
* The message consumption done synchronously
* Both need to be available in the same time
* Not decoupled in terms of time and space
* It relies on messages for establishing communication between the sender and the receiver

1. *Notifications:*

* Coupled in time and space
* Notifications send by clients to the server including callback arguments
* Notifications sent by server to the client including the result

1. *Message queuing:*

* Uses tuple-space, queues are provided with messages from producers and additional transactional, ordering and timing functionalities are provided by the message queue
* Same as shared space
  1. **Use Case diagr****am**

<<include>>

<<include>>

Subscribe module

Publisher

Subscriber

<<include>>

Functionality of subscribe service:

* Exposes method which an application (subscriber) can remotely invoke through a communication channel to subscribe events.
* Exposes method which an application (subscriber) can remotely invoke through a communication channel to unsubscribe events.
* Exposes method which an application (subscriber) can remotely invoke through a communication channel to renew events.

Functionality of subscriber:

* The subscriber application registers itself for one or more topic events.
* Receives events when the events are sent from the publisher.

Functionality of publisher:

* Sends event to the publisher service for publishing.

**5.5.1 Use case description**

**Use case name: Subscribe events**

Description: Allow interested user to subscribe events

Actor: Subscriber (interested users)

System: PC, mobile phone

Pre-condition: Subscriber already login into system.

|  |  |
| --- | --- |
| Actor action | System response |
|  | 1. Display all the subscriptions |
| 1. Choose events |  |
| 1. Select events |  |
| 1. Select First-fix (Not or non) |  |
| 1. Select second-fix |  |
| 1. Insert value |  |
| 1. Enter valid e-mail address |  |
| 1. Enter phone no |  |
| 1. Click “Subscribe” button |  |
|  | 1. Display confirmation message |
| 1. Click “okay” button |  |
|  | 1. Display successfully message |
|  | 1. Store subscription details into database |
|  | 1. Auto generates confirmation e-letter to subscriber’s email. |

Post-conditiPost-conditon: Subscriber will receive a confirmation message from the system that subscriber already subscribed an event.

**Use case name: Unsubscribe events**

Description: Allow subscriber to unsubscribe events

Actor: Subscriber

System: PC, mobile phone

Pre-condition: Subscriber already login into system.

|  |  |
| --- | --- |
| Actor action | System response |
|  | 1. Display all the subscriptions |
| 1. Choose events |  |
| 1. Select events |  |
|  | 1. Display “Unsubscribe” button |
| 1. Click “Unsubscribe” button |  |
|  | 1. Display confirmation message |
| 1. Click “okay” button |  |
|  | 1. Display successfully message |
|  | 1. Delete subscription details from database |
|  | 1. Auto generates confirmation e-letter to subscriber’s email. |

Post-condition: Subscriber will receive a confirmation message from the system that subscriber already unsubscribed an event.

**Use case name: Publish events**

Description: To allow publisher to publish events

Actor: Publisher

System: PC, mobile phone

Pre-condition: Subscriber already login into system.

|  |  |
| --- | --- |
| Actor action | System response |
|  | 1. Display all the publications. |
| 1. Choose events |  |
| 1. Select events |  |
|  | 1. Display “Publish” button |
| 1. Click “Publish” button |  |
|  | 1. Display confirmation message |
| 1. Click “okay” button |  |
|  | 1. Display successfully message |
|  | 1. Store publication details into database |
|  | 1. Auto generates confirmation e-letter to publisher’s email. |

PosPost-condition: Publisher will receive a confirmation message from the system that publisher already published an event.

**Use case name: Register an account**

Description: Allow subscriber or publisher register an account in order to subscribe or publish an event.

Actor: Publisher, Subscriber

System: PC, mobile phone

Pre-condition: Search the system available

|  |  |
| --- | --- |
| Actor action | System response |
|  | 1. Display user registration screen |
| 1. Enter publisher/subscribers’ details |  |
| 1. Click “Register” button |  |
|  | 1. Validate user inputs |
|  | 1. Display confirmation message |
| 1. Click “okay” button |  |
|  | 1. Store personal information into database |
|  | 1. Display successfully message |
|  | 1. Auto generates confirmation e-letter to publisher or subscriber’s email. |

PoPost-conPost-condition: Publisher or subscriber will receive a confirmation message from the system that publisher or subscriber already been registered.

* 1. **Database Design**

*5.6.1 Centralized database*

Le Subscribe (Fabret et al., 2001) and Xlyeme (Nguyen et al., 2001) are predicate-based message filtering systems that use centralized processing. In these systems, a predicate is a comparison between as attribute and a constant using relational operators as “>”, “<”, and “=”. The main issue they address is how to efficiently match an incoming event, in the form of attribute value pairs, with the predicates of a large number of queries. The key idea is to index predicates as well as cluster queries. In a particular, Le Subscribe uses multi-attribute hash indexes to evaluate several predicates in a query with a single operation. In addition, it groups queries based on number of contained predicates and the common conjunction of equality predicates, so many queries can be (partly) evaluated using a single operation. It further offers cost-based algorithms to find optimal clustering and to dynamically adjust it.

Publish –subscribe with expressions:

1. Storing expression as Table data
2. Metadata is needed

* To store information about values stored in the condition predicates
* Validate values stored when new or existing columns are modifies

1. Indexes can be added
2. Evaluating expressions

*5.6.2 NoSQL Database*

Since we have used NoSQL database to accomplish our system, consequently we have to describe what is NoSQL database exactly is, and what are the benefits of using NoSQL.

NoSQL stands for “Not Only SQL”. It is encompasses a wide variety of different database technologies that were developed in response to a rise in the volume of data stored about users, objects and products, the frequency in which this data is accessed, and performance and processing needs. On the other hand, for the relational databases, were not designed to cope with the scale and agility challenges that face modern applications, nor were they build to take advantage of the cheap storage and processing power available today.

**Different between NoSQL and SQL**

|  |  |  |
| --- | --- | --- |
|  | SQL Databases | NoSQL Databases |
| Types | One type (SQL DB) with minor variations | Many different including key-value stores, document databases, wide-column stores, and graph databases |
| Data Storage Model | >Individual records (e.g., “employees”) are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., “Supervisor”, “salary” etc.) much like spread sheet.  >Separate data types are stored in separate tables, and then joined together when more complex queries are executed. | Varies based on database type. E.g. key-value stores function similarly to SQL databases, but has only two columns (“key” and “value”), with more complex information sometimes stored within the “value” columns. |
| Scaling | Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required. | Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. |
| Data Manipulation | Specific language using select, insert and update statements. For example, UPDATE employee.id = 0565 WHERE employee.id = 0567 FROM employee | Through object-oriented APIs. |
| Consistency | Can be configured for strong consistency | Depends on product. Some provide strong consistency whereas other offer eventual consistency. |

*5.6.3 MongoDB*

MongoDB is one of the most popular document based NoSQL database as it stores data in JSON like documents. It is non-relational database with dynamic schema. It has been developed by the founders of DoubleClick, written in C++ and is currently being used by some companies like The New York Times, MTV Networks, etc.

In our system, we have implemented this database, besides the consideration of data manipulation, another reason behind was it is also open-source. Meaning to say that it can be downloaded implemented and scaled at little cost and development cycles are faster.

**Benefits of using NoSQL database**

* + - 1. *No schema or fixed data model*

Data can be inserted in a NoSQL database without first defining a rigid database schema. So the format or data model being inserted can be changed any time, without application disruption. This provides user the flexibility to evolve your data schema without modifying the existing data.

* + - 1. *Integrated Caching Facility*

In order to increase data output and performance advance NoSQL techniques cache data in system memory. This is in contrast to SQL database where this has to be done using separate infrastructure.

* + - 1. *Elastic Scaling*

It can expand transparently and horizontally to take advantage of new nodes, and they are usually designed with low-cost commodity hardware. Besides that, the servers can be added or removed from the data layer without application downtime.

* + - 1. *Speed*

For simple queries, it gives good performance, as all the related data are in single document which eliminates the join operations.

**Main references:**

MongoDB, Inc. 2015. ‘*Nosql database explained’*. Viewed on 8th June 2015. Available from: <<https://www.mongodb.com/nosql-explained>>.

Epik Theme. 2015. ‘*Nosql vs sql’*. Viewed on 8th June 2015. Available from: <https://blog.udemy.com/nosql-vs-sql-2/>.

Ramesh Natarajan. 2008. ‘*Sql vs nosql database differences explained with few example db’*. Available from: <http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/>.

* 1. **Class Diagram**

Defined by

**YahooStockEngine**

-observableCollection: Quotes

+YahooStockEngine()

+Fetch()

+Parse():observableCollection<yahooAPI>

+StorePublicationValue(stockname: String, stock value:String)

+updatePublicationValue(publishcationID: Guid, StockName:String, StockValue: String)

+GetDecimal(input: String)

+GetDateTime(input: String)

**SystemUI**

-length: Double

-width: Double

-layout: Integer

+setConnectionStatus()

+getConnectionStatus(): String

+getSubscriberInfor()

+notify()

1

1

inform

1

Informed by

monitors

1

Instructs

1

**Subscription**

-subscriptionID: Guid

-subscriber: MongoDBRef

-expression: GenericExpression

-notificationChannel: NotificationChannel

-extraElement: BsonDocument

+Event()

+setTitle()

+setCategory(): String

+setContent(): String

+getTitle(): String

+getCategory(): String

+getContent(): String

**subscriberProfile**

-userID: Guid

-username: String

-password: String

-extraElement: BsonDocument

+subscriberProfile()

+setSubscriberID()

+getSubscriber():String

+getPreferences(): String

+updateStatus(): String

+setSubscriberInfo(): String

+getSubscriberInfo(): String

+IsValid():Boolean

Monitored by

Instructed by

1

1..\*

**Subscribe**

-subscriberID:String

-subscriberName: String

-subscriberPhone:String

-subscriberEmail:String

+subscriberPreference()

**yahooAPI**

-LastTradePrice: decimal?

-Symbol: String

+yahooAPI( ticker: String)

+LastTradePrice(): lastTradePrice

+PropertyChanged: event PropertyChangedEventHandler

+ observableCollection: Quotes

1..\*

defines

subscribes

1

1..\*

Receive subscription

1

1..\*

notifies

notified by

This class diagram that represents the classes of the system implementation together with their attributes and operations (methods). The implementation of the system is covered by six classes as follows:

1. *Subscriber*

This class represents a bean that defines properties and behaviors for handling information of the subscriber entity.

1. *SubscriberProfile*

This class is a derivation from the super class profile. Added to this, it is a specialized bean that defines properties and behaviors for handling information pertinent subscriber profile.

1. *yahooAPI*

This class is responsible for notifying subscribers of published publication. It’s only activated when there is a publication to deliver and it is instructed by the event manager to do so.

1. *YahooStockEngine*

This class is responsible for initiating the communication between itself and the subscribe module UI class, and also instructs the notifier class to notify subscribers, and in case of undelivered events if direct them to the XML database for future retrieval.

1. *systemUI*

This class is responsible to let either subscriber or publisher to interact with user interface directly.

1. *Subscriptions*

This class is responsible to let subscriber to choose the topic they interested searching by title and content in order to receive subscriptions.

* 1. **Subscribe interaction sequence diagram**

**Subscription**

**Subscription Manager**

**Notification Service Provider**

**System UI**

**Subscriber**

Login()

CreateSubscription()

Subscribe()

NotifySubscribeUI()

SubscribeResponse()

<<create>>

Register()

Register()

NotifySubscribeUI()

ResponseRegistration()

**Publisher**

**System UI**

**Publication Manager**

**Notification Service Provider**

CreatePublication()

PublishResponse()

PublishEvent()

Login()

ResponseRegistration()

Register()

<<create>>

**Publication**

1. *System UI*

This class create a subscriber profile during subscription and store the created profile to the subscription information.

1. *SubscriptionManager*

This class has a set method to prepare a subscriber’s status, also manages subscription. It also directly interacts with the publicationManager.

1. *PublicationManager*

This class is a publication manager class that managers all published events by ensuring that events are channelled to interested subscribers, and also ensures safe storage of undelivered events in a persistent storage.

1. *NotificationServiceProvider*

This class accepts and disseminates events to subscribers, receiver’s subscriber details such as name, handphone, e-mail address and published events.

* 1. **Algorithms research framework**

Subscription\_new

Subscription\_ID

ArrayList

…

…

…

…

…

Signal:

*Every 20 new subscriptions*

*Every 5 minutes*

Publish module

When publisher publishing an event or more, every 20 new subscriptions will be received by subscribe module, then it will automatically send the signal to publish module. In addition, the subscribe module will send the signal to publish module every five minutes. No matter the subscribe module already received 20 new subscriptions and the time has not reached in 5 minutes’ time, it may still send the signal to publish module. Besides that, if the new subscription has not reached the specified quantities after 5 minutes, it may still send the signal to publish module. However, the subscription quantity has the high priority than timing.

**How to ensure message is correct?**

To ensure message is correct, store message in such a way that easy to read and organize is necessary. Let imagine that a table call “Subscription” in relational database and has “subscription ID”, “subscriber ID”, and “Interest Value (expression)” as attribute. There has another table call “Topic” and has “Topic ID”, “Topic Name (current value)”, “Content”. Therefore, clear picture will be look like Figure 5.2. When subscribers subscribe an interest a subscription will create for them and store their interest value. A message notification or warning will send to subscriber when current value (Content) reach a level that match interest of subscriber (interest\_value).

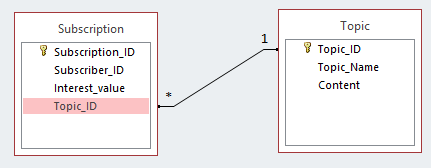


Figure 5.2: Relationship between subscription and topic

Assume record is stored in such a way that as show as Table 5.1 and Table 5.2.

Table 5.1 Subscription

|  |  |  |  |
| --- | --- | --- | --- |
| **Subscription\_ID** | **Subscriber\_ID** | **Interest\_value** | **Topic\_ID** |
| 1 | 1 | water>10|oxygen<5 | 1 |

Table 5.2 Topic

|  |  |  |
| --- | --- | --- |
| **Topic\_ID** | **Topic\_Name** | **Content** |
| 1 | Presence of water | water=11|oxygen=4 |

Now, observe the way of data store in table 5.1 and table 5.2. Is the data easy to be organized or analyzed by a system? The answer is “No!” This is because problem occurred when a system tries to retrieve value of “water” from “Content” column in “Topic” table. Since the data consist of multiple values, extra algorithms are needed to recognize the syntax of data in “Content” column and then perform looping to retrieve the value accordingly. To perform looping, an extra data attribute need to be stored in database. The attribute is referred to “number of attribute” and “name of attribute”. “Number of attribute” is used to control the round of looping (when the loop stop?) while “name of attribute’ is used as a key to retrieve the value of each attribute (where is the value?). To make it more clear, **Scenario 5.1 and Algorithm 5.1** is provided to show the step of how the algorithms retrieve the value.

**Scenario 5.1:**

Retrieve value of each attribute (water, oxygen) from the record as show in Table 5.3.

Table 5.3 target record of scenario

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Topic\_ID | Topic\_Name | Content | Number of attribute | Name of attribute |
| 1 | Presence of water | water=11|oxygen=4 | 2 | water|oxygen |

**Algorithm 5.1:**

This algorithm is referring to **Scenario 5.1**.

**Step 1:**

Retrieve data from database and put it into variable. Therefore, the output should be like diagram below.

String content = “water=11|oxgen=4”

Int number\_of\_attribute = 2

String name\_of\_attribute = “water|oxygen”

**Step 2:**

Split name\_of\_attribute into character and put it into a temporary list.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | a | t | e | r | | | o | x | y | g | e | n |

**Step 3:**

Run a loop to find index for each “|” in temporary list. For each loop, take out a character and compare with “|”. If the character is equal to “|”, then recording the index, else do nothing. The loop will stop when number of time which has found “|” is equal to “name\_of\_attribute minus one”. In the scenario in **Step 2**, index of “|” is 5.

**Step 4:**

Use the index which has found in **Step 3** to analyze some useful information as diagram below.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | a | t | e | r | | | o | x | y | g | e | n |

Length of temporary list- 1

Index of “|” + 1

0

Index of “|” – 1

**Step 5:**

Retrieve each key in temporary list base on useful information which has found in **Step 4**.

1st key = character from index 0 to 5(Index of “|” – 1) in temporary list

2nd key = character from index 6 (Index of “|” + 1) to 11(Length of temporary list – 1) in temporary list

**Step 6:**

Put the key which has found in **Step 5** into key list.

|  |  |
| --- | --- |
| water | oxygen |

**Step 7:**

For each key in key list follow **Step 8** to **Step 11.**

**Step 8:**

Split content into character and put it into a temporary list.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w | a | t | e | r | = | 1 | 1 | | | o | x | y | g | e | n | = | 4 |

**Step 9:**

Run a loop to find index for each “|” in temporary list. This step can refer to **Step 3.**

In the scenario in **Step 8**, index of “|” is 8.

**Step 10:**

Use the index which has found in **Step 9** to analyze some useful information as diagram below.

T = number\_of\_attribute

Index of 1st attribute start from 0 to (length of key in key list (**Step 6**) – 1)

Index of Nth attribute start from (index of (N-1)th “|” + 1) to (index of (N-1)th “|” + length of key in key list (**Step 6**)), while N = 2, 3, 4, 5, …, T

It is easy to compare (next step) after knowing the position (index) of each attribute in the list.

**Step 11:**

Retrieve the attribute in temporary list (**Step 8)** by using the information in **Step 10**. Then, compare the attribute with the key in key list (**Step 6**). If the attribute is equal with the key then retrieves the value, else do nothing. The formula to retrieve the value is shown in diagram below.

T = number\_of\_attribute

Index of value for Nth attribute start from (last Index of Nth attribute + 2) to (index of Nth “|” - 1), while N= 1,2,3,4, …, T-1

Index of value for Tth attribute start from (last Index of Tth attribute + 2) to (length of the temporary list (**Step 8**) – 1)

It is easy to retrieve the value of each attribute after knowing each position.

Base on scenario in **Step 8**, if the key is water, then the value should be 11. Else if the key is oxygen, then the value should be 4.

Algorithms 5.1 provide a simple and readable step to retrieve a value base on Scenario 5.1. Although, this algorithm able to solve the problem which has stated in Scenario 5.1 completely, but it is a bit tricky to implement. This is because the complicated step which has involved in the algorithms. The step of algorithms can become more complicate when illustrating it in more technique way. The reason behind is that some of the step will involve nested loop (loop within a loop).

For example about Step 5 of Algorithm 5.1, the outer loop is looped based on number of attribute. Then, 2 attribute mean loop 2 times. For each time of loop in outer loop, perform inner loop. The inner loop is looped base on the formula which has stated in Step 5 of Algorithm 5.1. So, the formula (1st key = character from index 0 to 5(Index of “|” – 1) in temporary list) will be used for first time of loop in outer loop. This formula “character form index 0 to 5” means it will loop 6 times in inner loop. For each time of loop in inner loop, retrieve a character base on each index. Next, the formula of 2nd Key will be used when second time of loop in outer loop. Then, continue with 3th, 4th, 5th, and so on until reach the end of number of time of loop in outer loop (reach max number of attribute).

Base on example above, it can be said that nested loop is needed for retrieving value which has stored in table 5.1 and table 5.2. Nested loop in this case need more carefulness to implement because a mistake may lead to a wrong value being retrieved. For instance, the loop should loop 3 times for retrieve value from index 0 to 2 in the list by default, what will happen if the loop only loops 1 times? (May be due to just a simply mistake in a formula like something minus one, but put three instead of one accidentally.) The answer is the information of value will inaccurate. Take an example of stock price, the value will become 5 instead of 500 when the loop only loops 1 time. This wrong value may cause an investor lose their profit. So, it is important that to ensure the control of loop is implementing correctly. The control of loop is referring to when the loop starts and when the loop stops. In this case, those formulas which has stated in Algorithm 5.1 play an important role in the control of loop. Unfortunately, this formula can be as complicate like the formula which has stated in Step 11 of Algorithm 5.1. For illustrate purpose, the formula is show again in Figure 5.2.

T = number\_of\_attribute

Index of value for Nth attribute start from (last Index of Nth attribute + 2) to (index of Nth “|” - 1), while N= 1,2,3,4, …, T-1

Index of value for Tth attribute start from (last Index of Tth attribute + 2) to (length of the temporary list (**Step 8**) – 1)

Figure 5.3 Formula to retrieve a value

Refer to Figure 5.3, the formula can be read like this: index of value for 1st attribute start from (last Index of 1st attribute + 2) to (index of 1st “|” - 1) when first time of loop in outer loop. In this case, the formula about index “from” (last Index of 1st attribute + 2) and index “to” (index of 1st “|” - 1) is useful to determine when the inner loop starts and when the inner loop stops. Another question will come when applying the formula of index “from” and index “to”. What is the value about those unknown variable (last Index of 1st attribute and index of 1st “|”)? To retrieve those unknown variable, it need another algorithms. This mean the algorithm needs another effort to implement. Therefore, the conclusion is the data which has stored in such a way that as table 5.1 and table 5.2 is difficult to organize because the retrieve process for the data is expensive and complicate.

Not only that, this complicate process may lead to system inefficiency in term of CPU usage. Notice that Algorithms 5.1 involve nested loop, what happen when the number of attribute increase? Assume that one time of loop consume one operation of CPU, then n \* m time of operation is needed for a nested loop, while n = number of time of loop in outer loop and m = number of time of loop in inner loop. This mean the number of operation in CPU will increase dramatically by just increase one in the problem size (i.e. m or n). On the other world, the CPU usage may become insufficient when the problem size is large.In context of measurement, the efficient of nested loop is O(n2) base on Big O notation. To make it more concrete, table 5.5 is provided to show overall efficiency of Algorithms. 5.1.

Table 5.4 overall efficiency of Algorithms 5.1

|  |  |
| --- | --- |
| **Step In Algorithm 5.1** | **Efficiency-time complexity (Big O notation)** |
| Step 1 | The efficiency in this step is highly depend on the algorithm which has encapsulated in SQL statement. Normally, loop will involve.  Big O notation: O(n). |
| Step 2 | The efficiency in this step is highly depend on the algorithm which has encapsulated in specify programing pre-define method such as “toCharArray(string pString)”. Normally, string data type is storing the data in Char array structure. Therefore, no loop is involve.  Big O notation: O(1) |
| Step 3 | Normal loop then loop through the Char array  Big O natation: O(n) |
| Step 4 | Index of each symbol “|” is stored in a list. Loop is needed to retrieve each index value for calculation. Therefore, number of time of loop in the loop is proportional to number of symbol “|” in the list (size of list)  Big O notation: O(n) |
| Step 5 | Nested loop  Number of time of loop in outer loop is proportional to number of attribute.  Number of time of loop in inner loop is proportional to length of attribute.  Big O notation: O(n2) |
| Step 6 | No loop is involved when put a key into key list if the size of list is enough.  Big O notation: O(1) |

|  |  |
| --- | --- |
| Step 7 | This step involve a loop which will repeat step 8 to step 11. Therefore, the total number of operation of CPU in this step is equal to (the sum of the number operation from step 8 to step 11) time (number of attribute).  Big O notation:  [O(1) + O(n) + O(n) + O(n2) + O(n2)][O(n2)]  Remove O(1) since the number of operation remain when problem size “n” (number of attribute) increase.  Since as “n” grow sufficiently, n2 > n. Therefore, O(n2) > O(n). So, choose O(n2) and ignore other because O(n2) will “dominate” the operation of CPU.  Finally, the big O notation for this step is [O(n2)][O(n2)] = O(n4) |
| Step 8 | The efficiency in this step is highly depend on the algorithm which has encapsulated in specify programing pre-define method such as “toCharArray(string pString)”. Normally, string data type is storing the data in Char array structure. Therefore, no loop is involve.  Big O notation: O(1) |
| Step 9 | Normal loop then loop through the Char array  Big O notation: O(n) |
| Step 10 | Index of each symbol “|” is stored in a list. Loop is needed to retrieve each index value for calculation. Therefore, number of time of loop in the loop is proportional to number of symbol “|” in the list (size of list)  Big O notation: O(n) |
| Step 11 | Two Nested loop  First nested loop (retrieve attribute):  Number of time of loop in outer loop is proportional to number of attribute.  Number of time of loop in inner loop is proportional to length of attribute.  Big O notation: O(n2)  Second nested loop (compare each attribute with each key. Then retrieve value of attribute if the attribute is equal with the key):  Number of time of loop in outer loop is proportional to number of attribute.  Number of time of loop in inner loop is proportional to length of value of attribute. (only loop when the attribute is equal with the key)  Big O notation: O(n2) |

Refer to table 5.4, notice that the highest value of big O notation is O (n4). If the problem size “n” is double, then the number of operation of CPU will octuplet. This result shows that Algorithm 5.1 is very inefficient in term of CPU usage.

Furthermore, The CPU usage will become even worst when “Interest value” match with “Content” (refer to table 5.1 and table 5.2). The purpose of this process is to find out which subscriber is interesting with current value. Note that “Content” is used for storing current value. To perform matching, the value of each attribute in both “Interest value” and “Content” is needed. Therefore, Algorithms 5.1 is necessary for retrieving value of each attribute from both “Interest value” and “Content”. This indicates that a system will use Algorithm 5.1 “two time” in order to retrieve both value of attribute for comparing. Using the algorithm for “two time” mean the system will double up the process of looping. “Double up” the process of looping may lead to double up the usage of resource of CPU. Finally, this make the system performance become “double” slow. In context of publish and subscribe module, “Slow” is a big issue because a delay of dissemination of message may kill thousands of people. For example, a delay of a warning message of fire may cause a victim late to escape from the fire. Eventually, the victim may die just due to a delay of warning message from publish and subscribe system.

In the nutshell, the way of store data as shown as Table 5.1 and Table 5.2 is insufficient. We should avoid implementing Algorithms 5.1 in system. So, it is better to store the attribute of interest event separately. In context of relational database, instead of store each value of all attribute in one column call “Content”, it is better to store each value in each column. This is because retrieve a value in a column is supported by SQL. Using SQL to retrieve a value is better than using the algorithm as show as Figure 5.3 definitely. The reason behind is that SQL is using CPU resource of database server which is separate from application server normally.

Chapter 6

Programming

# 6. Programming

In our system, we have used Microsoft Visual Studio 2013 programming tool to accomplish our project. Within this programming tool, we have chosen ASP.NET as program development for the whole system. It is because ASP.NET is a Microsoft Web application framework which built on Common language runtime for building dynamic web sites using one of the programming languages like C#, VB.NET etc. It supports 2 models which are Web Forms and ASP.NET MVC.

ASP.NET

MVC

Web Forms

However, we have chosen C# programming language and ASP.NET MVC as our first choice to accomplish our project.

* 1. **ASP.NET framework**

Figure 7 had shown the architecture of ASP.NET framework:

ASP.NET Core

ASP.NET Web Forms

ASP.NET MVC

Handlers

Many more..

Membership

Master Pages

Caching

Globalization

Security

Modules

Session

Microsoft .NET framework

ASP.NET Runtime

Figure 6.1: Architecture of ASP.NET framework

Since our project was musing ASP.NET MVC, so we will just further discuss the model of MVC and why we have chosen this MVC as our program development.

Basically, MVC is an architectural pattern which is has been around for sometimes now. Below are some terminologies included in MVC:

* + - 1. Patterns

In simple words Pattern is a solution to a problem in a context.

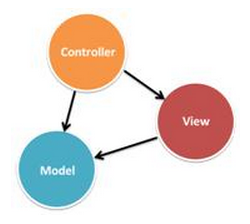
* + - 1. Architectural Patterns

It is something which solves our problem at sub system level or in short module level. It deals with the problem related to architecture of a project. It tells us how we can divide our system and especially why. We make Class libraries, Components, Web services to solve the problem.

* + - 1. MVC

It is an architectural pattern which let us develop an application having loosely coupling between each of these elements. The main intention behind MVC pattern is separation of concerns. It makes presentation or UI ignorant of business and user interaction logic. At the whole, MVC system should be divided as M – Model, V – View and C – Controller.

* Model – Is considered as smart and handles the Business rules, logic and data will be independent of other parts of MVC (controller and view).
* View – Is considered as dumb and is an output representation of model data. It may be an excel sheet, a web page showing list of records or just a simple text. View knows about only Model.
* Controller – It receives and dispatches the request in short it handles the user interaction and input logic. It knows about both model and view.



Generally, ASP.NET MVC is not replacing ASP.NET Web Forms. Both development models are existing and can be used to develop ASP.NET applications such as mobile app, etc. Nevertheless, both have pros and cons that we will discuss and compare here.

The summary of the high level comparison between Web Forms and MVC as shown in table (()).

Table 6.1: High level comparison between Web Forms and MVC

|  |  |  |
| --- | --- | --- |
| Features | Web Forms | MVC |
| Separation of concerns | No | Yes |
| Familiar Event Driven Model | Yes | No |
| ViewState Issues | Yes | No |
| Server Controls | Yes | No |
| Control over HTML | No | Yes |
| Test Driven Development | No | Yes |

**Advantages of using MVC**

Project Architecture

One of the advantages of using ASP.NET MVC is it enforces separation of concerns. Consequently there is very less chances of getting things more complex as shown on figure 8.

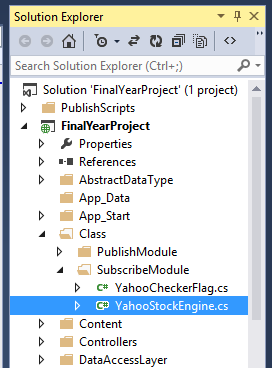


Figure 6.2: MVC Project Architecture

Performance

ASP.NET MVC don’t have support for view state, so there will not be any automatic state management which reduces the page size and so gain the performance.

Full control over HTML

ASP.NET MVC does not support server controls, only option available is using html input controls, so we will be sure about final html rendered at the end. We will also be aware about “id” of every element. And so integration of ASP.NET MVC application with third party JavaScript libraries like jQuery becomes easy.

Test Driven development and Reusability

In MVC controller is separate class so automatic testing is possible featuring Test Driven Development. Besides that, controllers are not bound to any specific view and so can be reused for multiple views.

Extensibility

ASP.NET MVC supports multiple view engines like aspx, razor and if required we can create our own.

**Disadvantages of using MVC**

1. More learning effort

Absence of event driven programming model and ViewState makes ASP.NET MVC a very difficult framework for developers with no or little experience in web application development.

1. Slow application development

It may need a lot of explanation during the application development. Rich server controls, event driven model and ViewState will decrease the development speed by great extent and developer will need to abstract from lots of the background complexities.

**Main References**

Web development tutorial. 2015. ‘Difference between ASP.NET webforms and ASP.NET MVC’ Viewed on 20 May 2015. Available from :< <http://www.webdevelopmenthelp.net/2013/10/difference-between-asp-net-webform-and-asp-net-mvc.html>>.

Marla Sukesh. 2014. ‘Web forms vs. MVC’. Viewed on 20 May 2015. Available from :< <http://www.codeproject.com/Articles/528117/WebForms-vs-MVC>>.

* 1. **Code analysis**

Some code had been abstracted that what we have implemented the critical path to link with third party systems.

If we need to call third party system’s application programming interface (API), we may need to invoke the method as following:

private const string BASE\_URL = “Third party URL”;

Once the API had been invoked, we need to fetch their latest quotes in order to get the current values. After fetched, the following sequence is to parse the latest values and store in our database.

The following code had shown our precedent sequence of invoking the Yahoo third party system.

**Method invode third party system >>**

public class YahooStockEngine

{

public YahooStockEngine() { }

private const string BASE\_URL = "http://query.yahooapis.com/v1/public/yql?q=select%20\*%20from%20yahoo.finance.quotes%20where%20symbol%20in%20({0})&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys";

public void Fetch(ObservableCollection<yahooAPI> quotes)

{

string symbolList = String.Join("%2C", quotes.Select(w => "%22" + w.Symbol + "%22").ToArray());

string url = string.Format(BASE\_URL, symbolList);

XDocument doc = XDocument.Load(url);

Parse(quotes, doc);

}

**Parse the latest values from third party system >>**

private void Parse(ObservableCollection<yahooAPI> quotes, XDocument doc)

{

XElement results = doc.Root.Element("results");

foreach (yahooAPI quote in quotes)

{

XElement q = results.Elements("quote").First(w => w.Attribute("symbol").Value == quote.Symbol);

quote.LastTradePrice = GetDecimal(q.Element("LastTradePriceOnly").Value);

IMongoQuery query = Query.EQ("topic.name",quote.Symbol);

PublicationDAL publicationDAL = new PublicationDAL();

List<Publication> foundList = publicationDAL.findPublication(query);

if (foundList.Any())

{

foreach (Publication element in foundList)

{

this.updatePublicationValue(element.publicationID, quote.Symbol, quote.LastTradePrice.ToString());

}

}

else

{

this.storePublicationValue(quote.Symbol, quote.LastTradePrice.ToString());

}

}

}

**Store the latest values from third party system >>**

private void storePublicationValue(string stockName, string stockValue)

{

UserDAL userDal = new UserDAL();

Publication newPublication = new Publication();

User publisher = new User();

IMongoQuery query = Query.EQ("user\_name", "Yahoo Finance");

List<User> userlist = userDal.findUser(query);

foreach (User user in userlist)

{

publisher.userID = user.userID;

}

MongoDBRef publisherRef = new MongoDBRef("UserCollection", publisher.userID);

newPublication.publisher = publisherRef;

StockPrice stockPrice = new StockPrice();

stockPrice.name = stockName; //"Dell"

stockPrice.price = stockValue; //"56.98"

newPublication.topic = stockPrice;

PublicationDAL dal = new PublicationDAL();

dal.createPublication(newPublication);

}

If the third party system had made some amendment and enhancement, we can still add the changes into our code as shown in below:

Quotes.Add(new yahooAPI("Add the new stock’s name here"));

However, the following had dedicated the yahoo model that can add new stock name within it.

public yahooModel()

{

Quotes = new ObservableCollection<yahooAPI>();

Quotes.Add(new yahooAPI("AAPL"));

Quotes.Add(new yahooAPI("MSFT"));

Quotes.Add(new yahooAPI("INTC"));

Quotes.Add(new yahooAPI("IBM"));

Quotes.Add(new yahooAPI("RVBD"));

Quotes.Add(new yahooAPI("AMZN"));

Quotes.Add(new yahooAPI("BIDU"));

Quotes.Add(new yahooAPI("SINA"));

Quotes.Add(new yahooAPI("NVDA"));

Quotes.Add(new yahooAPI("AMD"));

Quotes.Add(new yahooAPI("WMT"));

Quotes.Add(new yahooAPI("GLD"));

Quotes.Add(new yahooAPI("SLV"));

Quotes.Add(new yahooAPI("V"));

Quotes.Add(new yahooAPI("MCD"));

Quotes.Add(new yahooAPI("FAS"));

UserDAL userDAL = new UserDAL();

User publisher = new User();

IMongoQuery query = Query.EQ("user\_name", "Yahoo Finance");

List<User> userlist = userDAL.findUser(query);

## Chapter Summary and Evaluation

First and foremost, the main reason that we have chosen ASP.NET MVC model as our program development at the first place simply because it support for parallel development. In ASP.NET MVC layers are loosely coupled with each other, so one developer can work on controller, at the same time other on view and third developer on model. Besides that, by default ASP.NET is faster because of lack of viewstate and clean mark-up. But performance is subject and MVC by design is more performant that traditional ASP.NET webforms. Moreover, we want full control over the HTML that is rendered in the browser and we can afford the development time and overhead to do all of our own mark-up.

On the contrary, we have forgotten to consider the timeline of the project’s dateline. We thought we have enough time to accomplish this project, but unfortunately we are not familiar with designing MVC applications. Moreover, it required more learning efforts and we have not much time to create efficient unit test for our user interface without the overhead of the entire managed runtime.

In the nutshell, we have burn mid-night to study the concept of ASP.NET MVC model before we started to code our program. However, my team mate and I have done a lot of research from internet and gained a lot of the particular knowledge to accomplish our system.

Chapter 7

Software Testing

# Software Testing

In this section we evaluate the performance of the ontology publish/subscribe system (OPS) with a variety of stimulated workloads. In the meantime, in this series of experiments we assume that the publish/subscribe system is subject to a large number of subscriptions, that these subscriptions stay in the system for a long time, and that the system must handle a high rate of events. These are the basic assumptions upon which we designed the matching algorithms. Besides that, this also represents the key requirements under which, that we assume our system will have to operate. However, in these experiments we measured the event throughput, the memory usage and the subscription loading time of our system using the different matching algorithms.

**7.1 Software Testing Method**

The prototype system was implemented in C#, and the performance tests discussed below were performed on a common Notebook PC with an Intel Core i7 4500U CPU at 1.8GHz and 4GB RAM running Windows 8.1.

To demonstrate the efficiency of our algorithm, we have compared the performance of the two algorithms under the same environment and workloads. We have used the Apache JMeter and build-in performance monitor tools as our experiment testing. Apart from that, for the simplicity for user, we call our algorithm as the event-matching algorithm and their algorithm as OPS algorithm in the follows.

At first we have defined a parameter matching rate, meaning the ratio of matched subscriptions to total subscriptions for a given event. The value of matching rate considerably affects the performance of graph matching algorithms. In figure ((2004fund10)), ((2004fund20)) and ((2004fund30)), its show the matching times of the event-matching algorithm under different numbers of subscriptions. In the experiments, the value of A is 10, B is 20 and C is 30, and the value of S varies from 5 – 100. The resulting matching rates are always around 5%. From the figures we can see that the matching time is just 1.5, 1.9 and 2.7 seconds respectively when the number of subscriptions is 30.

Table (()) shows the comparison of the CPU usage and memory usage of two algorithms. From the table we can see the event-matching algorithm is much faster in event matching than the OPS algorithm. When there are just 30 subscriptions, the matching time of the OPS algorithm reaches -0.989%, while the CPU usage of the event-matching algorithm is merely -24.746%. Therefore, the conventional graph matching algorithms are not suitable for the publish/subscribe systems where there are large numbers of subscriptions.

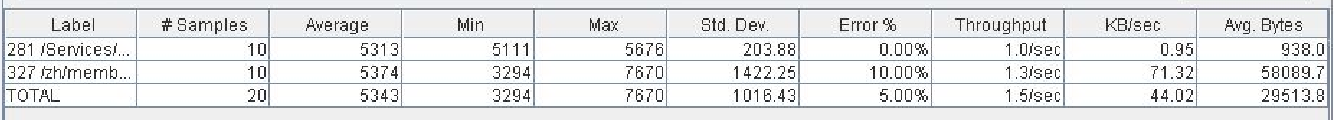
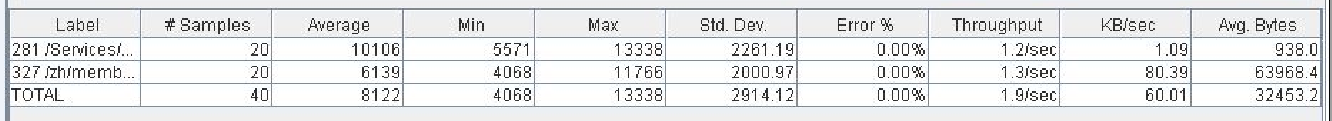
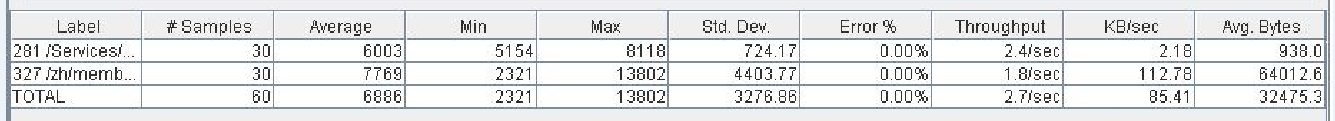
Eventually, the performance of event-matching algorithm is more efficient than OPS algorithm. Moreover, although some of the event-matching algorithm test result has lower performance compared to OPS algorithm, however, the CPU usage for that event-matching algorithm test is also lower compared to OPS algorithm.

**7.2 Results**

Figure 7.1: OPS algorithm throughput for 10 subscriptions

Figure 7.2: OPS algorithm throughput for 20 subscriptions

Figure 7.3: OPS algorithm throughput for 30 subscriptions

****

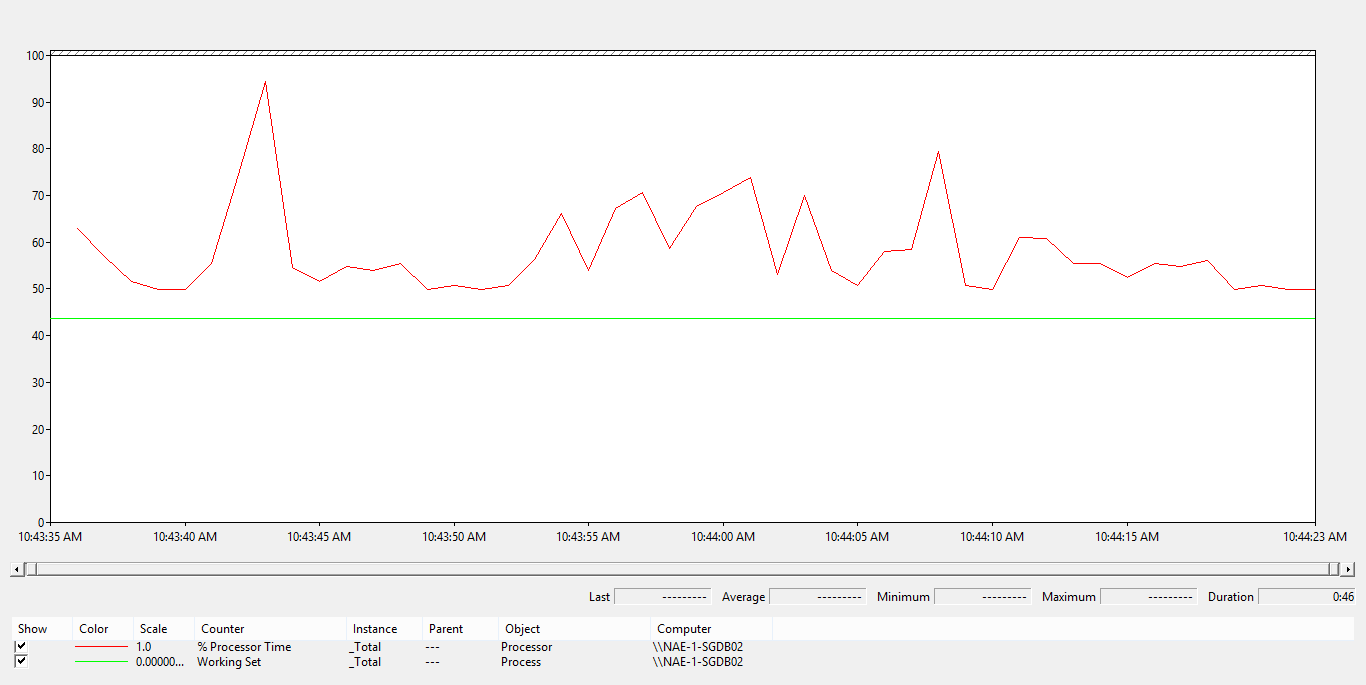
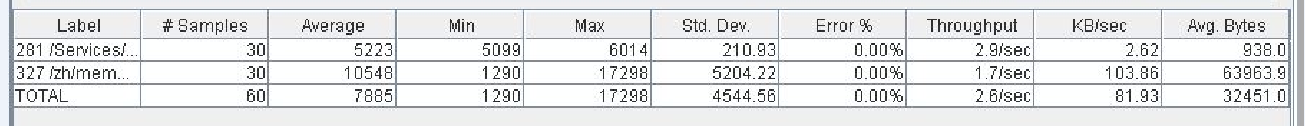
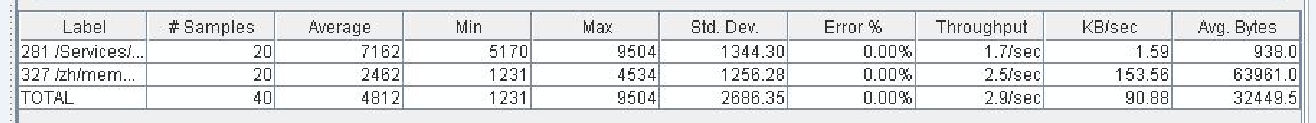
****

Figure 7.7: CPU usage and Memory usage for 10 subscriptions (Event-matching algorithm)

Figure 7.2: Event-matching algorithm throughput for 30 subscriptions

Figure 3.5: Event-matching algorithm throughput for 20 subscriptions

Figure 7.4: Event-matching algorithm throughput for 10 subscriptions

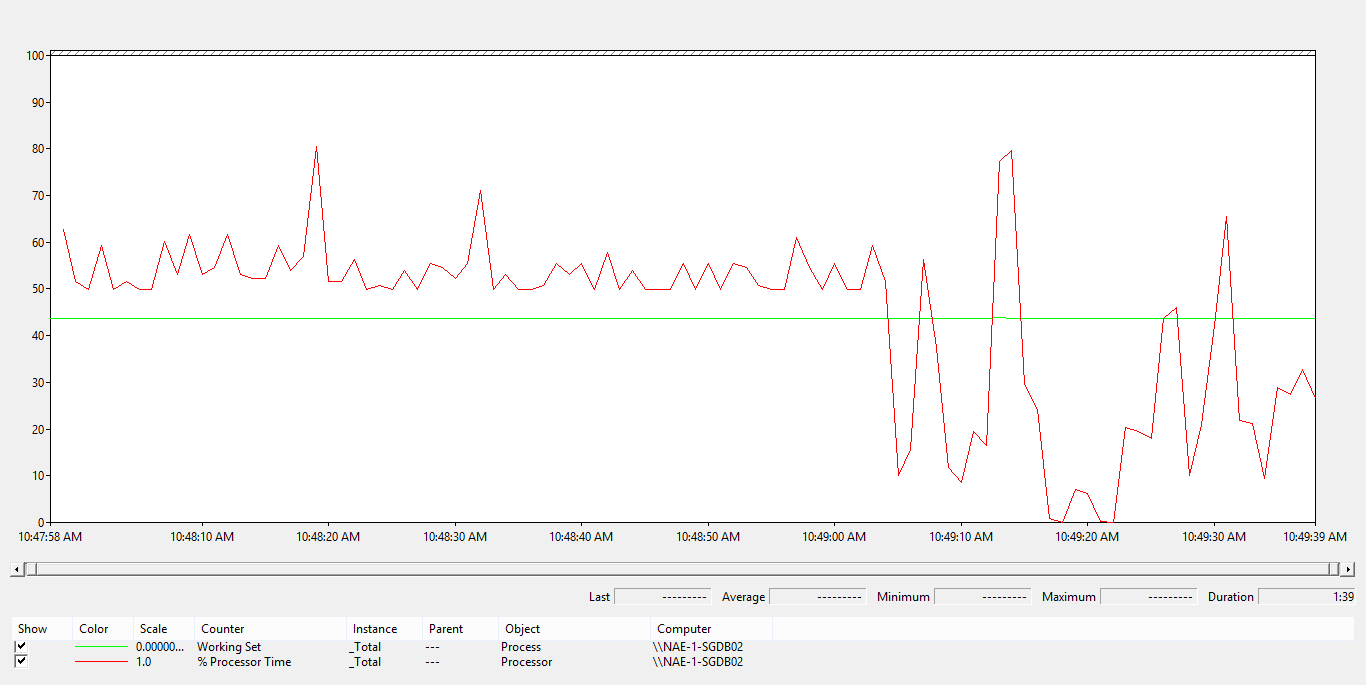
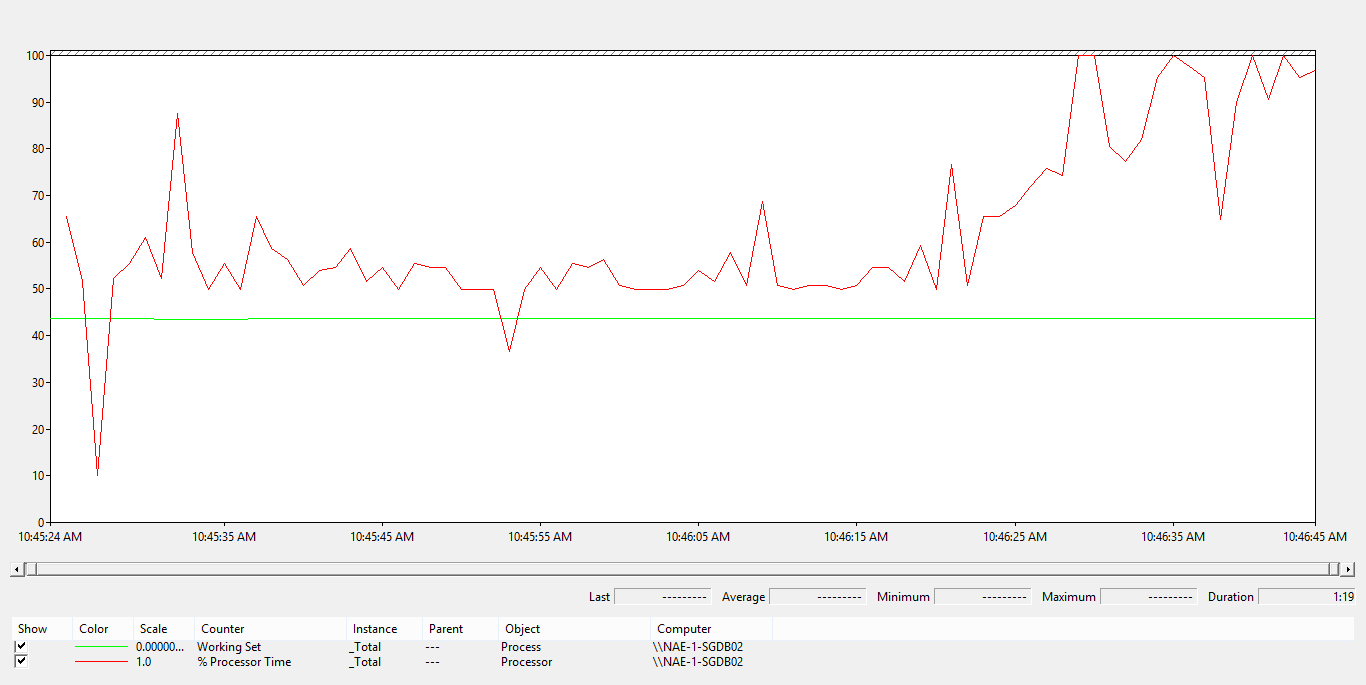
****

Figure 7.4: CPU usage and Memory usage for 30 subscriptions (Event-matching algorithm)

Figure 7.8: CPU usage and Memory usage for 20 subscriptions (Event-matching algorithm)

Table 7.1: Comparison of the CPU usage and memory usage between two algorithms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithms: | | Subscriptions | Subscriptions | Subscriptions |
| Concurent subscribers: | | 10 | 20 | 30 |
| Jmeter | Start Time (Jmeter) | 10:43 | 10:45 | 10:48 |
| End Time (Jmeter) | 10:45 | 10:47 | 10:50 |
| Successful Login (%)\*\* | 100 | 100 | 100 |
| Error (%) | 0 | 0 | 0 |
| Event-matching algorithm Throughput | 1.1/sec | 2.9/sec | 2.6/sec |
| OPS algorithm Throughput (Threads/Expected Average response Time) | 1.5/sec | 1.9/sec | 2.7/sec |
| Event-matching algorithm Average Response Time (secs) | 7.612 | 4.812 | 7.885 |
| OPS algorithm Average Response Time (secs) | 5.343 | 8.122 | 6.886 |
| Server (Local PC) | CPU Usage (%) - Before run Jmeter | 54.458 | 61.907 | 53.638 |
| CPU Usage (%) - Jmeter completed | 50.186 | 86.977 | 28.892 |
| **CPU Usage (%) - Differences (completed minus before)** | -4.272 | 25.07 | -24.746 |
| **CPU Usage (%) - Differences (completed minus before) -** OPS algorithm | -3.634 | -1.138 | -0.989 |
| Memory Usage - Before run Jmeter | 4377364187 | 4358710886 | 4361468928 |
| Memory Usage - Jmeter completed | 4366023885 | 4377128960 | 4375764173 |
| **Memory Usage - Differences (completed minus before)** | -11340302 | 18418074 | 14295245 |
| **Memory Usage - Differences (completed minus before) -** OPS algorithm | -11352942 | 323686 | 6932821 |

**7.3 Chapter Summary and Evaluation**

This chapter has outlined the experiment plan within this study. The main problem we have faced at the beginning stage was we can’t even test the result for our proposed system due to our system is uncompleted. However, the solutions we have found were asked our senior at our internship’s company. Nevertheless, after described our problem to our corresponding internship company’s seniors, our code can be run successfully and they have taught us how to test the efficient of our system by using various testing tools such as Apache JMeter and build-in performance monitor. Those seniors were willing to teach us patiently and guide us step-by-step that how to use the corresponding testing tools to test the efficient of our program.

Honestly without those helpful seniors assistance, our system may considered failed and cannot do any software test plan or experiment plan until now. On the whole, we are managing to generate the efficient of our proposed algorithm compare to other existing algorithms in our experiment plan.

Chapter 8

Conclusion

1. **Conclusion**

In this paper we propose a main memory algorithm for filtering event contents with respect to conjunctions of constant, attribute and comparison operator like “>”, “<”, “=” predicates. Our algorithm uses a schema based clustering strategy in order to minimize the number of subscription checks. While subscription clusters are accessed through multi-attribute hash tables. We also propose a dynamic algorithm to create and remove clusters and hash tables dynamically when the event patterns have changed or set of subscriptions is modified. Besides that, the possibility and necessity measures are expressive; these two matching degrees could be used for different optimistic and pessimistic users. Furthermore, the algorithm is designed with respect to conjunctions of predicates, but it can be easily extended to disjunctions as long as we substitute the minimum operator with other Boolean combination functions like maximum. On the whole, we also think that our algorithm can be used as efficient pre-filtering modules in more powerful publish/subscribe systems such as SQL triggers and continuous queries.

**8.1 Summary**

Much has been written about real-time publish/subscribe systems, but little effort has been expended in documenting the patterns, optimization and architectures required to implement publish/subscribe models in component –based software architectures. Besides that, there is very little or no empirical evidence to support the performance and predictability claims of several of these systems when used in component-based systems, even when research concentrates on real-time applications. We have addressed some of these weaknesses in this project.

**8.2 Completeness of the project**

So far there has been very little research in the complex and difficult are of failed assessments. This project has made an important start, but there is a long way to go. Nevertheless we have involved our proposed main algorithm within our project, yet somehow MVC framework had restricted many uses of functionalities and there are lack resources available via internet, consequently, our project was uncompleted due to those particular reasons. Nevertheless, what we have researched and proposed this paper, we strongly believe that our main contribution of our work is that it greatly improves the expressiveness of the publish/subscribe system and at the same time it keeps a high matching efficiency.

**8.3 Contributions**

We had proposed a dynamic algorithm to create and remove clusters and hash tables dynamically when the set of subscriptions is modified. It is because due to insertions and deletions or when event patterns are changing. However, the main advantage of our system is the unique combination of the high expressiveness of content-based filters and scalability and fault tolerance of a system.

**8.4 Future Improvements**

So far there has been very little research in the complex and difficult are of failed assessments. This project has made an important start, but there is a long way to go. This project is based on software yet it can also can extended to the networking environment by adopting the same proposed message delivery model for publish/subscribe communication paradigm for content dissemination. This would enable the researcher to identify some algorithm that would be tested with the following metrics, minimal processing load, minimal bandwidth consumption, network jamming and notification delay to solve the performance of the publish/subscribe systems in a dynamic and generic environment especially with mobile clients by addressing the issues concerning networking (network jamming and overloading). Furthermore, it can also compare with varying the number of messages sent then also records the effect of increasing the number of users of messages sent.

**8.5 Issues and solutions**

First of foremost, we did really face a lot of problems while doing this project and this paper which included technical issues, project management issues, team dynamics problems, etc. The most critical issue that we have faced was technical issues during our system implementation. Conscientiously, we have chosen the wrong software development tool at the beginning stage although we have proposed another development tool in our software requirement specification. One of the reasons was we wished to learn something new from this project, so we have chosen ASP.NET MVC 5 as our main software development tool instead of ASP.NET web form. Another reason was MVC is considered as a simple paradigm and it can avoid mixing code from the three categories into one class. For example, because the view doesn’t care about the underlying model, supporting multiple file formats is easier, just needed to add a model subclass for each. However, although these two development tools are under the same programming, which is C# programming language, yet the concept behind and implementation’s methods are unlike to each other, especially the controller in MVC, it can only send commands to its associated view to change the view’s presentation of the model. Therefore, the complexity is high to develop the applications using this pattern. It also increases the event-driven nature of the user interface code, which can become more difficult to debug.

Secondly, about the project management issue, we have burn midnight for certain period due to we have a lot assignments need to be accomplished. The main reason behind was our course’ program structure was unstable and keep changing the requirement due we are the first badge Bachelor Degree in TAR UC. The managements were also messy and students can’t manage well their time management due to unstable causes. Consequently, this influence may lead to insufficient project management issues. However, after we have get used to it for those exhausting period, we have gained the experience that every situation has to be prepared the worst plan and the best cases in order not to be so toilsome.

To summarize all, I can conclude that I have learnt a lot of new knowledge within this project and paper. Not only to fulfill the requirement of TAR UC for this paper, yet also challenge myself to learn new knowledge. Moreover, while doing for this project and paper, I have gained various experiences that I ever met such as well manage on time management, how to discuss with team mate with effective way and sufficient manner, etc. On the whole, our project will be slightly trend to disadvantage due to our developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. But once again, the main advantage of our system is the unique combination of the high expressiveness of content-based filters and scalability and fault tolerance of a system.

# 9. References/Bibliography

Arens, Y., and C. A. 1992. ‘Proceedings of the 1st international conference on information and knowledge management’. *Planning and reformulating queries for semantically-modeled multi-database systems.* Pp 99 – 101.

Benjamin, E., Kuziemsky, C., Peyton, L., Middleton, G. and Mouttham, A. 2010. *Policy-based data integration for e-health monitoring processes in a B2B environment: experiences from Canada* [online]. Canada, Ottawa University. Viewed on 23 October 2014. Available from: < <http://www.scielo.cl/scielo.php?pid=S0718-18762010000100006&script=sci_arttext> >.

Benjamin, N., Serge, A., Gregory, C. and Mihai, P. 2001. ‘Monitoring XML data on the web’. Sigmod record (ACM special interest group on management of data). 30(2), pp.437-448.

Christine, C., Michael, N.R., and Sheri W.M. 2001. ‘IEEE Computer’. Resource integration using a large knowledge base in carnot. December 2001, pp 55 – 62.

Edward. S, Michael. S, and Arnon, R. 2004. ‘ACM transactions on database systems’. *Using semantic values to facilitate interoperability among heterogeneous information systems.* 19(2), pp. 254-290.

Eidson, B., Maron, J., Pavlik, G. and Raheja, R. 2005. ‘Proceedings of the first international workshop on design of service-oriented application’. *SOA and the future of application development.* Amsterdam.

Fabret,F., Jacobsen,A., Dennis,F. and Pereira,J. 2001. *Filtering algorithms and implementation for very fast publish/subscribe systems* [online]. California, Santa Barbara. Viewed on 16 July 2014. Available from:<<http://msrg.org/publications/pdf_files/2001/FaJL01-Filtering_Algorithms_and_Implementatio.pdf>>.

Fabret,F., Jacobsen,A., Dennis,F. and Pereira,J. 2001. *Filtering algorithms and implementation for very fast publish/subscribe systems* [online]. California, Santa Barbara. Viewed on 16 July 2014. Available from:< <http://msrg.org/publications/pdf_files/2001/FaJL01-Filtering_Algorithms_and_Implementatio.pdf> >.

Huhns, M.N. and Singh, M.P. 2005. ‘IEEE internet computing’. *Service-oriented computing: key concept and principles*. 9(1), pp. 75-81.

Massimo, P., Takahiro, K., Terry, R.P., and Katia, S. 2012. ‘The 1st international semantic web conference’. *Semantic matching of web services capabilities*.

Milenko, P., Ioana, B., Hans-Arno, J. 2003. *Semantic Toronto Publish/Subscribe System* [online]. Berlin, Germany. Viewed on 16 July 2014. Available from:<<http://www.vldb.org/conf/2003/papers/S36P06.pdf>>.

Mitchell, J.D., Siegel, M.L., Schiefelbein, C.S., Babikyan, A.P. 2007. *Applying publish-subscribe to communication-on-the-move node control* [online]. U.S.A, Wideband Tactical Networking group. Viewed on 22 September 2014. Available from: < <https://www.ll.mit.edu/publications/journal/pdf/vol16_no2/16_2_10Mitchell.pdf> >.

Petrus, S.S. 2007. *Guaranteed real-time delivery of context-aware messages in publish/subscribe systems* [online]. Agriculture, University of Zululand. Available from: < <http://uzspace.uzulu.ac.za/bitstream/handle/10530/105/Guaranteed+real-time+delivery+of+context-aware+messages+-+P.S.+Shabangu.pdf;jsessionid=AA5E06986F87C8BA0D00B378F56DE836?sequence=1>>.

Peyton, L., Hu, J., Doshi, C. and Seguin, P. 2007. ‘8th world congress on the management of eBusiness’. *Addressing privacy in a federated identify management network for e-health.* Toronto

Peyton, L., Hu, J., Doshi, C. and Seguin, P. 2007. ‘8th world congress on the management of eBusiness’. *Addressing privacy in a federated identify management network for e-health.* Toronto

Tarkoma, S. 2012. *Publish/Subscribe systems design and principals*. Viewed on 22 September 2014. Available from: <<http://www.amazon.com/reader/1119951542?_encoding=UTF8&page=36>>.

Wikibooks. 2011. *C sharp programming: delegates and events.* Viewed on 15 July 2014. Available from < <http://en.wikibooks.org/wiki/C_Sharp_Programming/Delegates_and_Events> >.

**10. User Guide**

This section is provided a clear instruction that allow users to install and use the prototype.

## 10.1 System Document

System document is provided the instruction to setup the prototype.

* + 1. **System (hardware and software requirements).**

In order to use the prototype, user must install the Microsoft Visual studio Ultimate 2013 and MongoDB. The prefer platform is Window 7/8 with 64 bit.

* + 1. **Installation**

To install MongoDB:-

* User must go to the setup folder in the DVD and find a folder call “MongoDB”. Then copy the file into “C:\” directory.

For window 7: -

* Go to window > All programs > Accessories > Command prompt
* Bear in mind that before enter the command prompt directory, user may need to right click the command prompt and click “run as administrator” button.

For window 8: -

* Press “window + Q”. A search window will pop out.
* Then, type “cmd” and find command prompt
* Bear in mind that before click the search button (enter), user may need to right click the command prompt and click “run as administrator” button.
* Click “Yes” in the user account control dialog box.
* Command prompt will be shown.

After opened the command prompt:-

* Firstly, go to the setup file and find a text file call “comment”. Then copy the command inside the text file and paste it into command prompt. Next, press “enter” to create a window service for MongoDB.
* Secondly, type “net start MongoDB” in the command prompt and press “enter” to start the window service. A successful message as Figure 10.1 had shown below.



Figure 10.1 Successful Message

To use MongoDB command:-

* At command prompt enter “cd c:/MongoDB/bin” and press “enter”.
* Then, enter “mongo” and press “enter’. A message as Figure 12.2 will show.



Figure 10.2 Connect to DB message

* Now, user able to use MongoDB command in the command prompt already.
* The MongoDB command is showed in the table 12.1.

Table 10.1 MongoDB command

|  |  |
| --- | --- |
| **Command** | **Definition** |
| show dbs | To show all database |
| use [databse name] | To use the specify database |
| db | To show current database |
| db.getCollectionNames() | To show all collection name in current database |
| db.[collection name].find().pretty() | To show all record in specify collection |
| exit | To close MongoDB |

## In order to ensure the MongoDB started successfully, user can go to Task Manager > Services panel to double confirm that MongoDB had already been run as shown in figure 10.3.

Figure 10.3: MongDB is inside the Task Manager

## To install Microsoft Visual studio Ultimate 2013:

* User must go to the setup folder in DVD and find a folder call “Visual studio 2013”. Then copy the folder into desktop.
* Go into the folder, and then double click “vs\_ultimate”. A setup window will prompt out.
* Follow the instruction to install it.
  1. **Operation Document**

This section provides a brief step-by-step guide on how to use the installed software and how to run the system.

1. Go to setup folder and copy “FinalYearProject” folder into local drive. Normally user may copy into desktop.
2. Open the Microsoft Visual Studio 2013 Ultimate.
3. Open the “FinalYearProject” solution file as shown in figure 10.4.

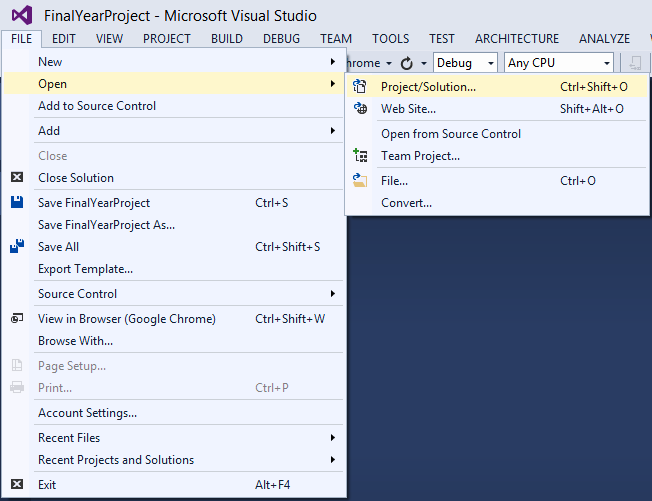


Figure 10.4: Open Project

1. After opened the solution file, user may able to view all the class files under the solution explorer panel as shown in figure 10.5.



Figure 10.5: All class file under the solution explorer

1. Before run the whole system, user must:
   1. Make sure internet connection is available (will grab the real-time values from yahoo stock price API).
   2. User may need to insert their G-mail account and password as shown in figure 10.6. No other SMTP service providers are allowed such as hotmail, yahoo, live, outlook, sina, etc. The directory is located at **FinalYearProject solution file >** **Class folder > PublishModule folder > SendEmailCommand.cs.**

If the user didn’t have their existing G-mail account, user may need to create a new account at Google corresponding website.

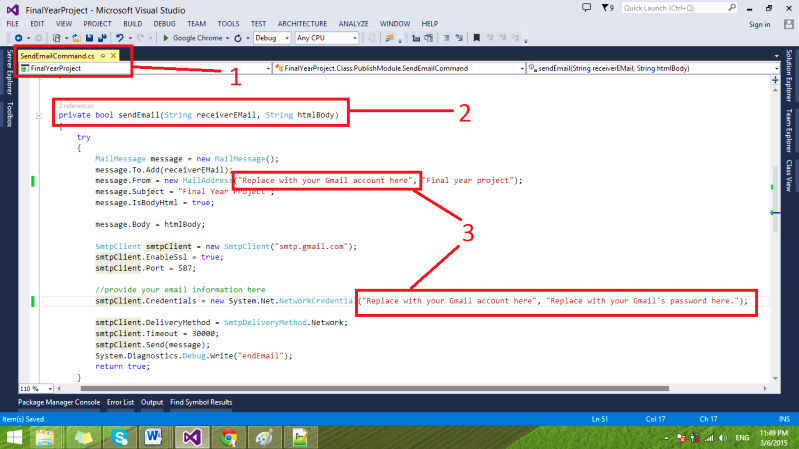


Figure 10.6: Insert Gmail account and password

1. After the user has fulfilled the requirement on step 5, user may direct click the run button or press (Ctrl + F5) to start the whole system as shown in figure 10.7.

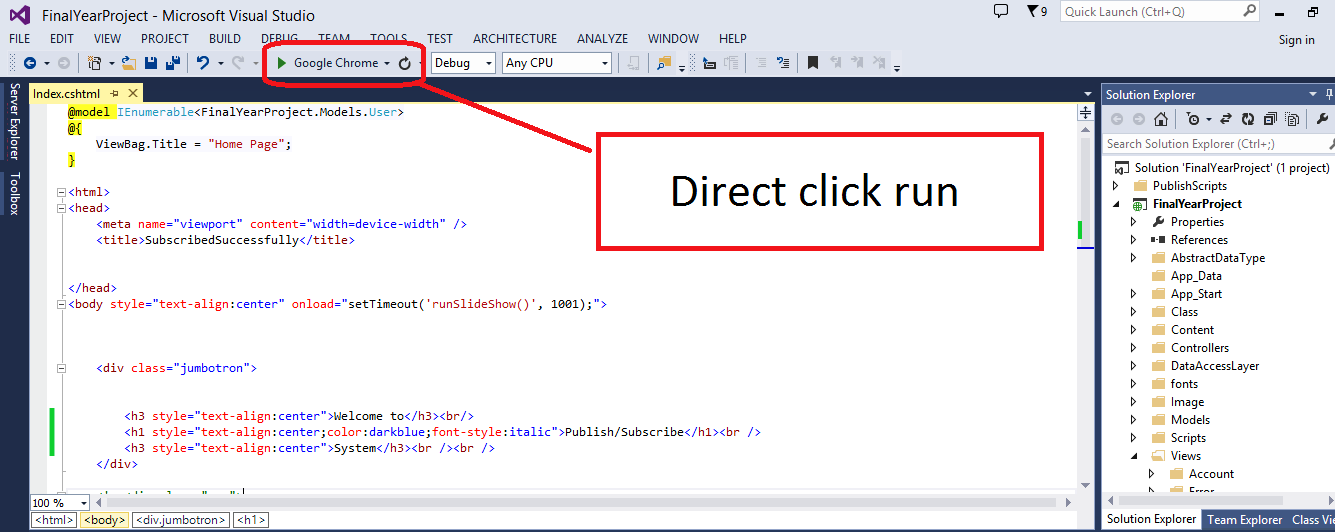


Figure 10.7: Click run button

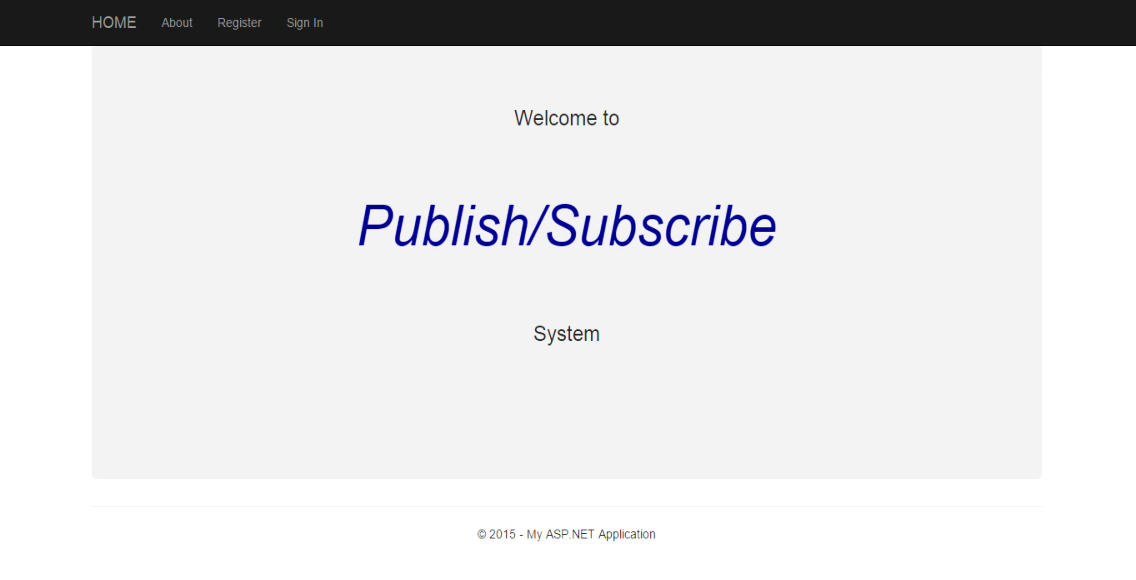
1. After the user has run the project, it will redirect to the main page of the system at user’s default browser as shown in figure 10.8.

Figure 10.8: Main Page of publish/subscribe system

1. New user is allowed to register a new account as shown in figure 10.9 and there are no validations required.

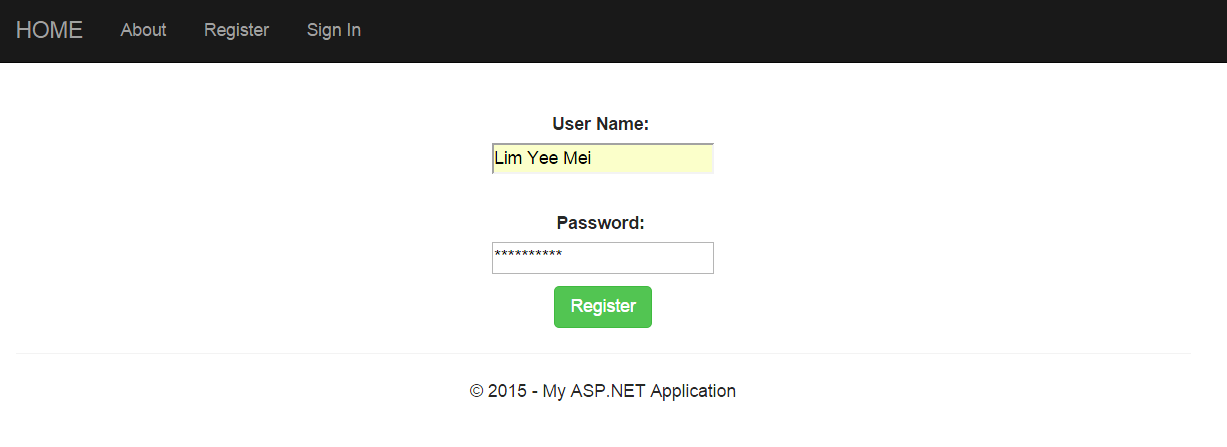


Figure 10.9: User Registration

1. After registered, user can sign in with username that user has registered previously as shown in figure 10.10.

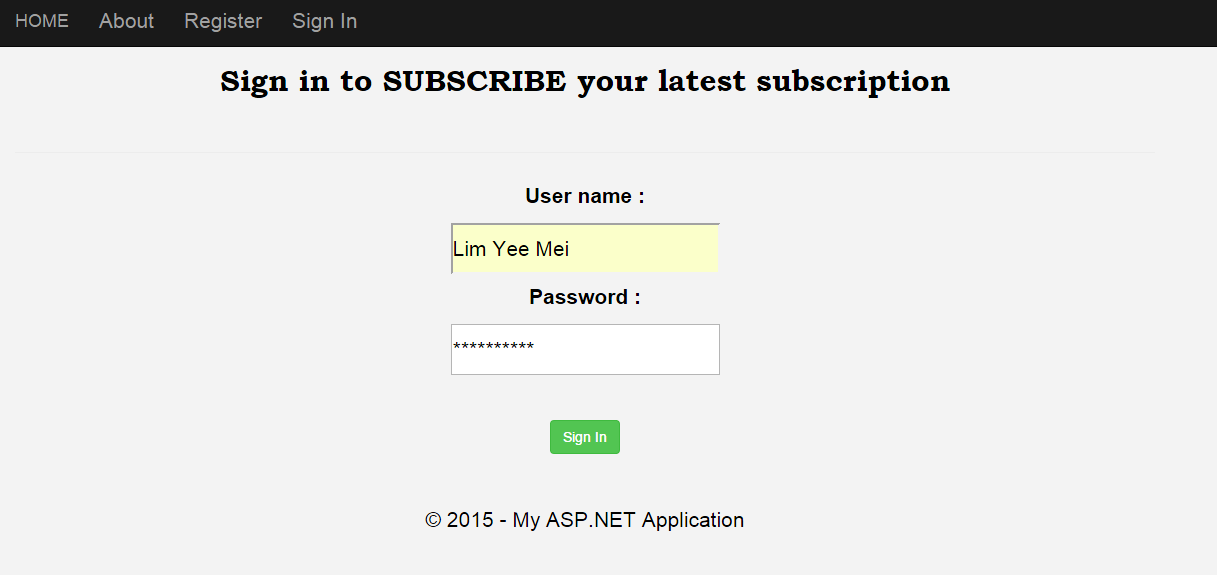


Figure 10.10: User login

1. The main page of the whole system will be shown as figure 10.11. So far our system is only available for stock price subscription.

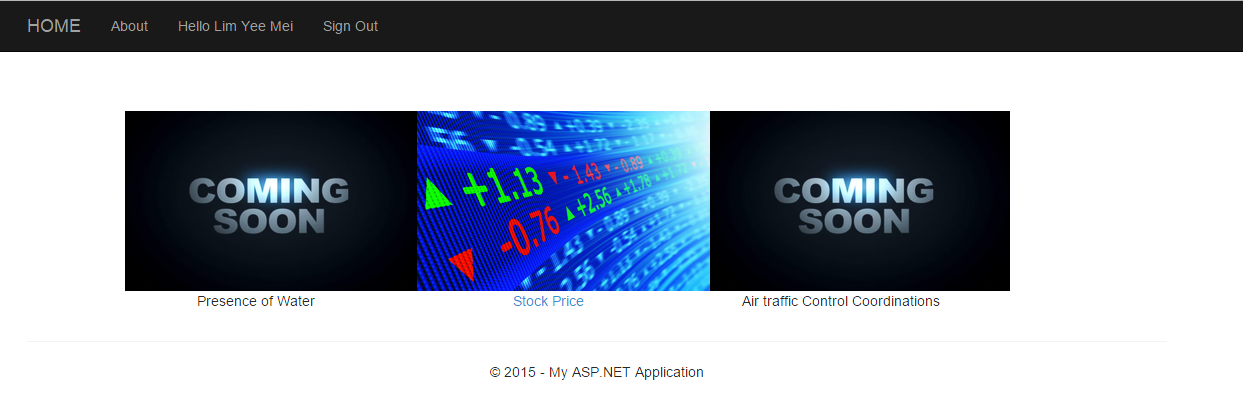


Figure 10.11: After login page

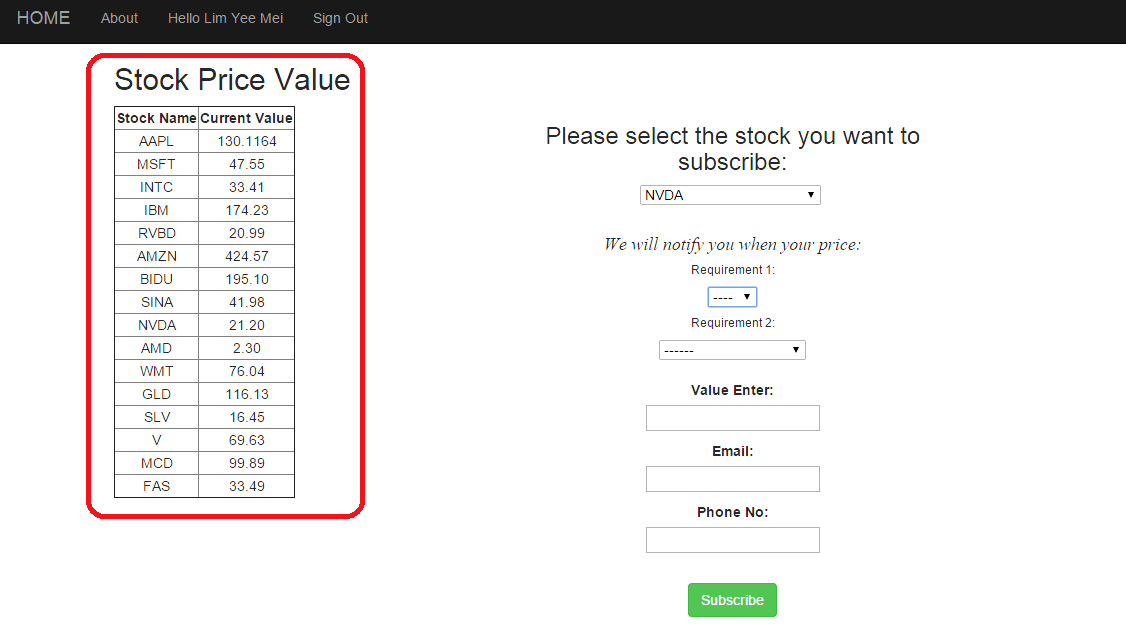
1. Figure 10.12 had shown the subscription content to subscribers. Once the user has filled up the particular fields, user will receives the subscriptions via email and phone number.

Figure 10.12: Subscription page

Select from upward to downward

Real-time values

1. After user has subscribed the interested event, the successfully message and thank you page will be shown as figure 10.13. User only allows subscribing one interested event at a time. If user is interested more than one event, user can click “click here” button to subscribe another interested event.

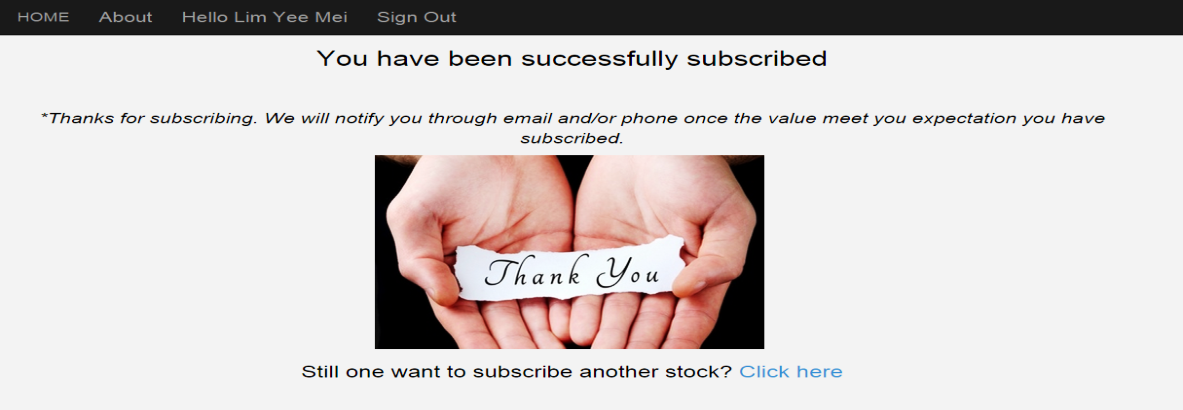


Figure 10.13: Thank you page after subscriber subscribed

1. Sign out button is available if user wishes to sign out. The subscription that subscribers subscribed has been successfully sent to subscriber’s email.

**11. Appendices**

**A.1 Other Existing Algorithms**

Mapping Functions

Synonyms

Concept Hierarchy

Subscriptions

Matching Algorithm

Figure 11.1: S-ToPSS System Architecture

New event from concept hierarchy

Original Event

New event from mapping function

“Root” event

“Root” subscription

Original subscription

Figure 11.1 shows the S-ToPSS system architecture. When a new event or subscriptions arrives, the synonym transformation is always done first in order to rewrite the event or subscription using “root” attributes. For each new event, the concept hierarch stage may create additional events. The same is true for the mapping function and concept hierarchy stages can be executed multiple times. The reason is for this is that the concept hierarchy stage can create new events for which additional mapping functions exist and vice versa.

**A.2 Data Structures and Algorithms**

The matching algorithm proceeds in two stages. First predicates are matched and, second is matching subscriptions identified. This is a similar break-down as applied in many crisp matching algorithms.

A.2.1 Data Structure

Predicate evaluation is based on two data structures: a hash table to index predicates according to their names and a predicate vector to store the degree of match for each predicate. Subscription evaluation is based on the list linked to each predicate to record the subscriptions that contain it and a subscription vector to keep track of the degree of match of each subscription. The overall data structure is described in figure 11.2.

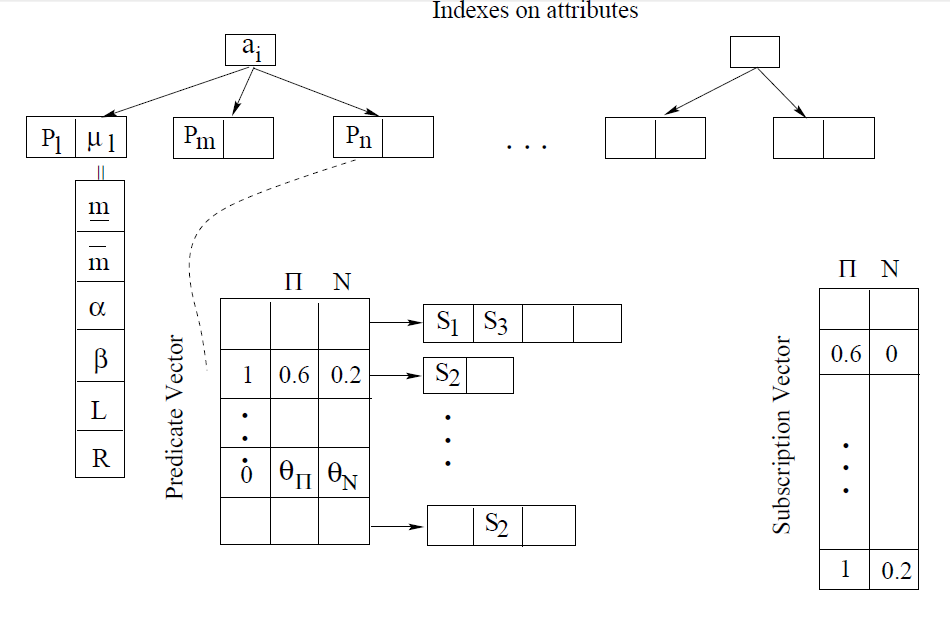


Figure 11.2: Indexes on attributes

**A.3 Corona**

Corona is a topic-based publish/subscribe system for the Web. It provides asynchronous update notifications to clients while interoperating with the current pull-based architecture of the Web. Users register their interest in some Web content by providing its URL and receive updates asynchronously about changes posted to that URL.

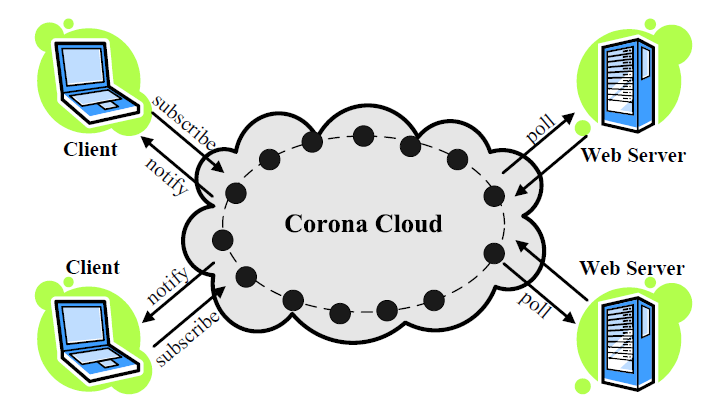


Figure 11.3: Corona Architecture

* Is a distributed publish/subscribe system for the Web.
* It detects Web updates by polling cooperatively and notifies clients through instant messaging.

**A.4 Publish/Subscribe with database integration**

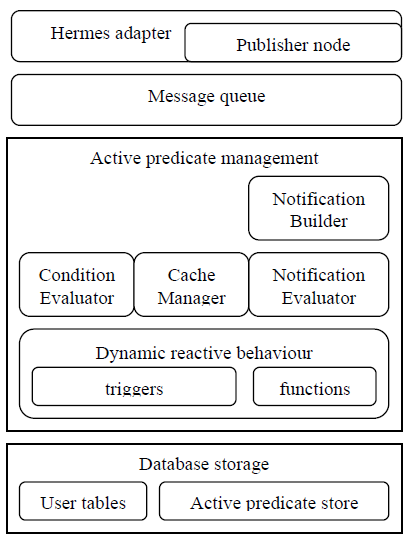
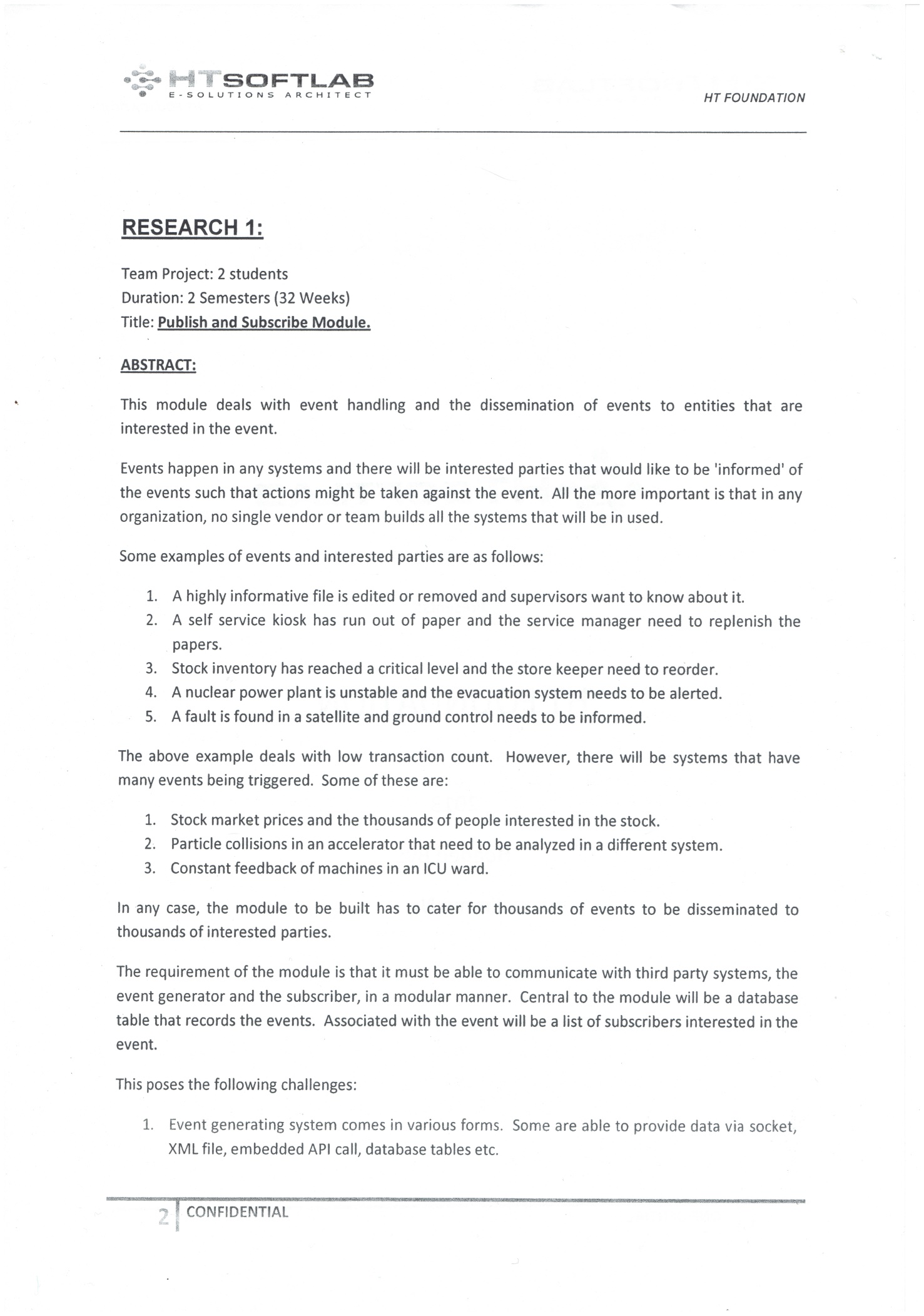
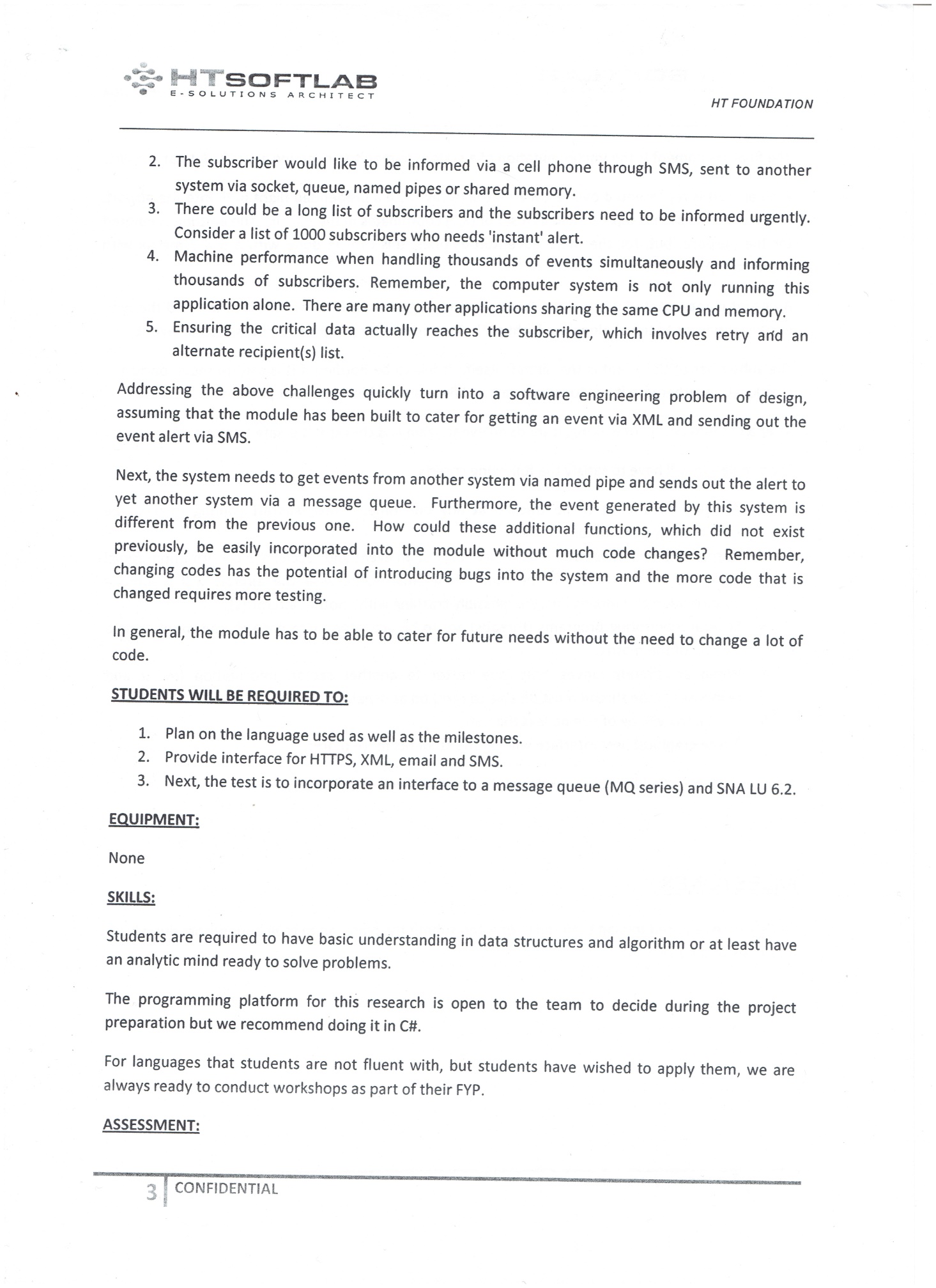


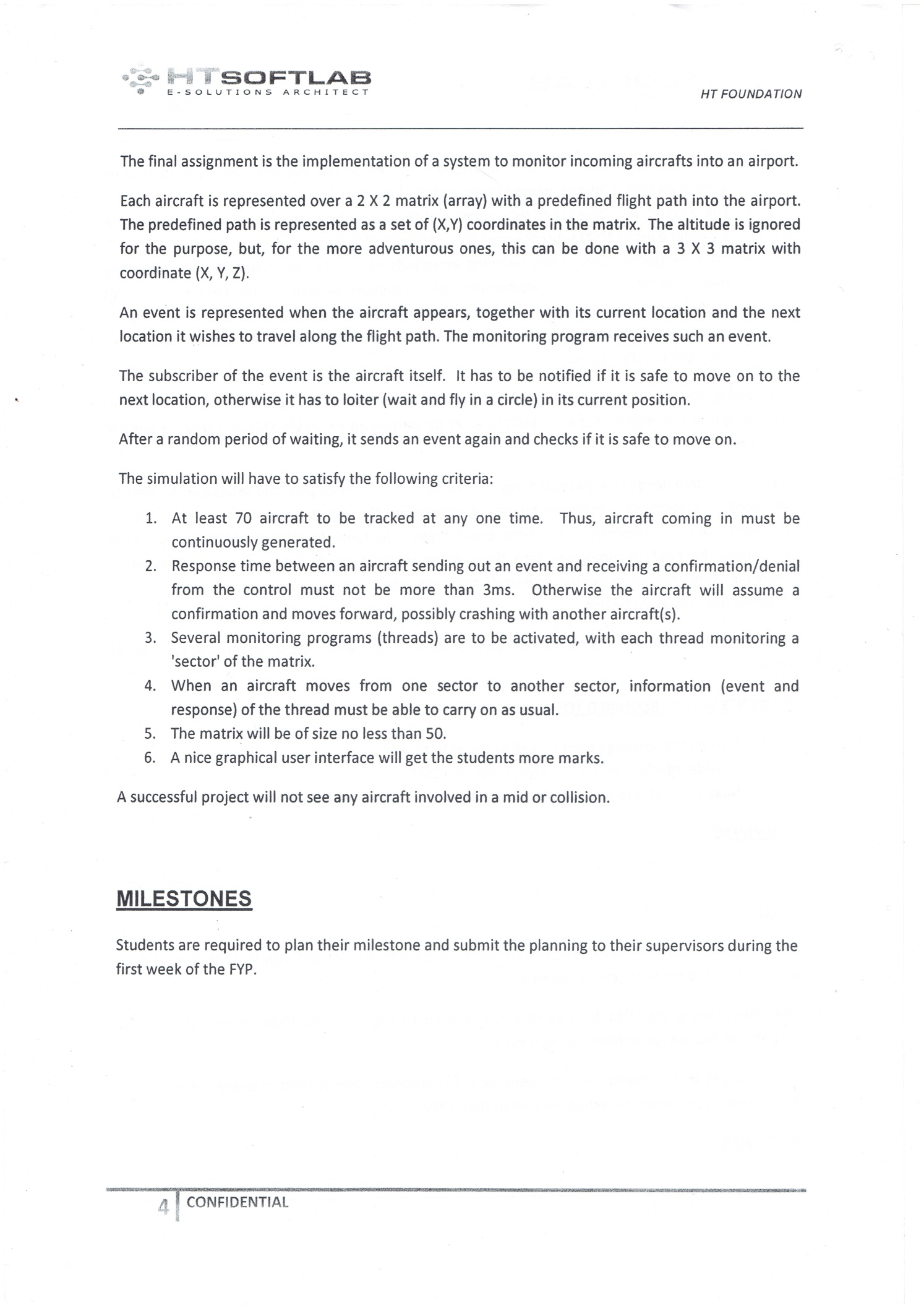
Figure 11.4: Integrated Database and publish/subscribe system architecture

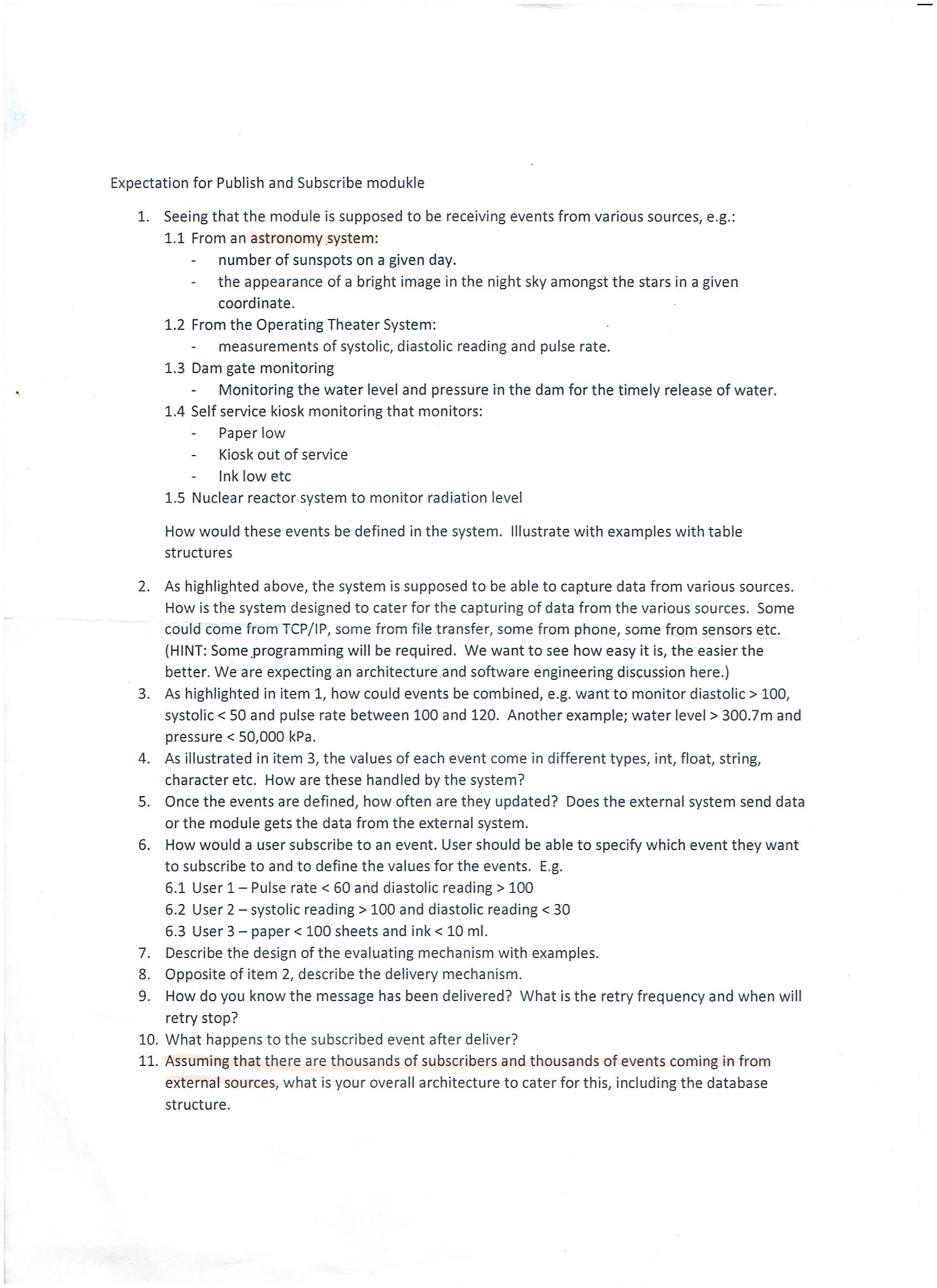
Figure 11.4 gives an architectural perspective of the various components involved in the integration. It is our publish/subscribe communication system, to form one global event-base system.

**B. HT Foundation Company’s Requirements**









**C. Experiment Test Plan and Result**

|  |  |
| --- | --- |
| **Student Details** | |
| **Name** | **Tan Ze Ming, Wang I - Sheng** |
| **Programme** | **RSF3** |
| **Project Title** | **Publish and Subscribe Module Dynamic Event Subscription** |

|  |
| --- |
| **Hypothesis** |
| Our program still remains very smooth even though there are 1000 processes. |
| **Materials and Apparatus** |
| * Source Code * A laptop * A hand phone * Apache JMeter tool * Build-in Performance Monitor tool |
| **Steps/Procedures** |
| 1. Open Performance monitor 2. Go to Data Collector Set > User Defined > Create New Data Collector Set 3. Select Create manually (Advanced) > create data logs (select Performance Counter) 4. Set performance counter to 1 seconds 5. Monitor the % processor time (total, add) 6. Monitor process, monitor server database 7. Select Save & Close radio button option 8. Start doing running JMeter script for different transaction in local PC 9. Ensure all transaction are shown in database |
| **Variables** |
| Independent variable : Username  Dependent variable : Time taken to receive email notification  controlled variables : Subscriber’s email, type of message, type of computer |
| **Recording of Data** |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | Algorithms: | | Subscriptions | Subscriptions | Subscriptions | | Concurent subscribers: | | 10 | 20 | 30 | | Jmeter | Start Time (Jmeter) | 10:43 | 10:45 | 10:48 | | End Time (Jmeter) | 10:45 | 10:47 | 10:50 | | Successful Login (%)\*\* | 100 | 100 | 100 | | Error (%) | 0 | 0 | 0 | | Event-matching algorithm Throughput | 1.1/sec | 2.9/sec | 2.6/sec | | OPS algorithm Throughput (Threads/Expected Average response Time) | 1.5/sec | 1.9/sec | 2.7/sec | | Event-matching algorithm Average Response Time (secs) | 7.612 | 4.812 | 7.885 | | OPS algorithm Average Response Time (secs) | 5.343 | 8.122 | 6.886 | | Server (Local PC) | CPU Usage (%) - Before run Jmeter | 54.458 | 61.907 | 53.638 | | CPU Usage (%) - Jmeter completed | 50.186 | 86.977 | 28.892 | | **CPU Usage (%) - Differences (completed minus before)** | -4.272 | 25.07 | -24.746 | | **CPU Usage (%) - Differences (completed minus before) -** OPS algorithm | -3.634 | -1.138 | -0.989 | | Memory Usage - Before run Jmeter | 4377364187 | 4358710886 | 4361468928 | | Memory Usage - Jmeter completed | 4366023885 | 4377128960 | 4375764173 | | **Memory Usage - Differences (completed minus before)** | -11340302 | 18418074 | 14295245 | | **Memory Usage - Differences (completed minus before) -** OPS algorithm | -11352942 | 323686 | 6932821 | |

This page is intentionally left blank to indicate the back cover. Ensure that the back cover is black in color.