# Functional Requirements And Software Architecture Specification
# COS 301 Team Alpha Project
# Version 2.0

Amy Lochner 14038600
Avinash Singh 14043778
Christiaan Nel 14029368
Christiaan Saaiman 12059138
Gerard van Wyk 14101263
Marc Antel 12026973
Themba Mbhele 14007950

GitHub *https://github.com/AvinashSingh786/COS301-Alpha.git*

University Of Pretoria

March 2016

# Contents

# 1 Introduction

This document defines the Software Requirements Specification and Technology Neutral Process Design for COS 301 Team Alpha. The Computer Science Department has expressed a need for the creation of a system, which can allow researchers to keep track of publications which they are currently working on, or have already published.
The aim of this project is to follow a structured software development process in order to produce a product which provides the client with all the required functionality in an elegantly designed and built product. A collaborative and co-operative approach is required between all stakeholders who are involved in this project.
The information, specifications, and diagrams within this document are presented in order to provide testable requirements which correlate to the client's needs.

# 2 Vision

The client for this project, Ms. Vreda Pieterse who is a member of the Department of Computer Science at the University of Pretoria, has solicited us to develop a system. The purpose of this system is to record and oversee all publications of staff members or research groups, within the Department of Computer Science. The system will assist the Head of Department to track the progress on any papers which are in the process of being written; as well as determining whether or not a staff member is under performing thus allowing the Head of Department the opportunity to provide advice to these staff members.

# 3 Background

## 3.1 The client's problem

The client has solicited us to develop a system that has specific usability goals, as well as certain user-experience goals. Currently, the Department of Computer Science does not have a system with which to monitor the progress of staff member's publications, nor to keep track of how many publications an author is working on. Furthermore, it is necessary to determine which publications will be presented at different conferences, as well as the reporting capabilities, and the ability to remind users of deadlines. The system should provide all these requirements in a secure, flexible and intuitive manner.

## 3.2 Future business/research opportunities

It is desired that this system will encourage authors to collaborate with other authors on similar topics and to expand the users base knowledge of ongoing research projects.

# 4 Software Architecture Documentation ...

This document defines

## 4.1 Architecture requirements

In this section extract the architectural requirements from the software requirements, including:

### 4.1.1 Architectural Scope

Architectural responsibilities that need to be addressed by the software architecture are as follows:

- Providing a persistent infrastructure - being a database stored on a dedicated server
- Providing a reporting infrastructure
- Providing an infrastructure for process execution
- Providing a backup infrastructure (e.g. making use of RAID)
- Providing a user interface through which the user can interact with the website/app
- Providing a means of Session management
- Providing a method of prevention against SQL injections
- Provide a log in system
- Provide a means with which to query the database in an interactive way
- Provide a means by which users can be notified about reminders

### 4.1.2 Quality requirements

- Performance

  The performance of a system refers to the behaviour of the system in terms of repsonse time and throughput. Changes and modifications to the system should be done in the most cost and time effective manner to provide a good user experience. One method of ensuring the performance is visible to the user is to make use of feedback tools to ensure the user is aware that the system is performing as expected or to notify the users should something have gone wrong.

- Security

  Minimising the possibility of leaking information, maintaining data integrity, and avoiding session hijacking. This is very important as personal information will be stored in the database(e.g. email). The system will also have different user types with different access permissions hence security will be needed to ensure that a specific user only has the capabilities

afforded to them.

We will achieve security by making use of 3 methods, namely: prevention, detection, and recovery. We will ensure that we have put means in place through which we may be able to detect any threats to the system, for example validating any information that is required to be sent to the server. Preventing threats to the system can be enforced through limiting the access channels, making use of authentication, not allowing any external sources to be accessed through the system, etc. Recovery will be achieved through cancelling requests and maintaining a working back up state to reroll changes.

- Maintainability

  This refers to how well the system can be modified to accommodate new functionality, access channels, fix bugs and improve the performance of the system. As it is likely that multiple people will be working on the system the code has to be easy to read and understand. We will achieve this by commenting our code to allow for a quick study of the code. We will employ a means by which to allow pluggability of our code to ensure that large amounts of code will not be needed to be changed/removed.

  The system should not only be maintainable in terms of issues with the system but it should also be maintainable in terms of usability, i.e user permissions should be allowed to be changed, etc.

- Scalability

  Scalabilty refers to the system's ability to handle increased traffic or workload. This will be implemented by ensuring that users can make the same requests simultaneously.

  We will need to ensure that resources are managed wisely in order to avoid lost updates, uncommitted data and inconsistent retrievals.

- Reliability

  As the system is required to support a fairly large user base it should allow for effective and safe concurrent use of the system. The system should ensure that no users can perform tasks that their user type does not afford to them.

  As access channels are via a web browser/android app it is essential that these access channels always have access to the server and thus database and ensure that the connection is stable, reliable and safe at all times. The system should be maintained in order to ensure that the system can perform all the required tasks in an effective and efficient manner. We will achieve this through unit testing, making use of concurrent resource locks and eliminating single points of failure.

- Auditability

A requirement of the system is that all actions or changes to the should be logged to enable users with the appropriate permissions to be able to view all actions or changes to the system, who they were made by and when. This is essential in ensuring that the system can be rolled back to a stable state should something undesirable happen.

We will implement this by having a log file running at all times which makes use of timestamping to indicate when changes were made, the system should allow to be rolled back to a stable state should it be deemed neccessary. ACID Properties will be applied to ensure that the database maintains a stable, reliable and current state.

- Integrability

  Will the system be able to integrate with other technologies.

- Usability

  Usability will be achieved through the implementation of an intuitive, easy to use, easy to understand and an aesthetically appealing interface through which the user will be able to perform all system functionality afforded to them through their user type.

  We will ensure that the user interface performs all tasks in the most efficient and direct means. The interface should not cause the user any irritation in the form of colour schemes, delayed functionality, over complication of tasks.

## 4.2 Integration and access channel requirements

The CS Research System will serve as the basis of the entire system, possibly integrating with future systems. Integration can be made possible through the use of a RESTful API to enable cross-platform concurrent communication through HTTP or HTTPS. By using a service-oriented architecture and thereby componentizing the system, other systems can more easily integrate through service calls to these different components. A web interface as well as a mobile application, specifically on Android, should be made available for the end users. Other platforms should then be able to communicate with the system through simple REST API calls. Additionally, services such as Google Calendar and Google Mail can also be integrated into the system through simple API calls made available by Google.

### 4.2.1 Architectural constraints ...

It is desired that this system will encourage authors to collaborate with other authors on similar topics and to expand the users base knowledge of ongoing research projects.

# 5 Architectural patterns or styles

Architectural Patterns are generally reusable solutions to commonly occuring problems within a given context. Similar to software design patterns, architectural patterns are larger in scope though. The architectural patterns address various problems such as:

- Software Engineering
- Computer Hardware limitations
- High availablility
- Minimization of Business Risk

For this project and the scope thereof, I suggest either one of two architectures are used or possibly both.

## 5.1 Model-View-Controller

Model-View-Controller, here on out referred to as MVC, is popular architecture used for implementing systems with user-interfaces. Later this architecture became extremely popular for designing web applications and for this reason I suggest using this architecture. This architecture is easy to implement and to maintain. It is divided into three seperate departments:

- The Model is in charge of managing the data, logic and rules of the system.
- The view is responsible for outputting the representation of the data that the user perceives.
- The controller takes in input and converts it to commands for the model view.

Current day MVC architecture has moved away from thin client coding due to technologies such as AngularJS, EmberJS, JavaScriptMVC and Backbone.

## 5.2 Service-orientated Architecture

A service-oriented architecture (SOA) is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any vendor, product or technology.

A service is a self-contained unit of functionality, such as retrieving an online bank statement. By that definition, a service is an operation that may be discretely invoked. However, in the Web Services Description Language (WSDL), a service is an interface definition that may list several discrete services/operations. And elsewhere, the term service is used for a component that is encapsulated behind an interface. This widespread ambiguity is reflected in what follows.

Services can be combined to provide the functionality of a large software application. SOA makes it easier for software components on computers connected over a network to cooperate. Every computer can run any number of services,

and each service is built in a way that ensures that the service can exchange information with any other service in the network without human interaction and without the need to make changes to the underlying program itself.

# 6 Architectural tactics or strategies

In this section, we will discuss certain strategies that will be used to achieve the quality requirements stated in section 4.2 and were further expanded upon in section 5.1.2.

- Hashing and Salting Passwords

  To ensure that sensitive information of the users of the system is not leaked in the event of a database compromise, all passwords will not be stored in plain text. Instead, all passwords will be hashed with a strong one-way hashing algorithm, such as the sha256, so that in the event of a database compromise, it will be near impossible to compute the actual passwords from the hashes. To make the hashes more effective, we will add random data, known as salt, to the passwords before hashing them. This will make it extremely difficult for intruders to use pre-hashed tables to perform a look up to find a matching password.

- Prepared Statements

  To prevent the illegal extraction of data and/or the destruction of data from the database through SQL injection and blind SQL injection attacks, we will make use of prepared statements. Prepared statements also have other benefits other than security. Even though prepared statements execute a statement multiple time, they reduce parsing time as the preparation of the query is only done once. This will improve the efficiency and performance of the system. Also, because prepared statements bind values to parameters, the bandwidth on the server is minimized because you only need to send the parameters to the server and not the whole query. This also improves the performance of the system.

- Caching

  Because the system relies heavenly on databases, there will be a significant number of access attempts to the the databases. This has the potential of creating a bottleneck that gets worse as the number of concurrent users increases. This means that the system will scale poorly and the performance of the system will decrease drastically. The best way to combat this is to limit accesses to the databases were possible. To accomplish this, we will make use of a distributed cache system. A distributed cache will particularly be effective in read operations and will provide large performance gains as it reduces the processing times of applications and it limits database accesses. This will improve the scalability of the system drastically and thus will accommodate a large number of concurrent users.

- Database Normalization

  To make the system flexible, we will create the database tables and link them to one another were it is appropriate to do so according to rules that protect the data and makes it flexible. This process is known as normalization. Normalization eliminates redundancies and inconsistent dependencies. This makes it simple to makes changes to the system and also makes it easy to add new components to the system.

# 7 Use of reference architectures and frameworks ...

...

# 8 Access and integration channels

### 8.0.1 Apache CouchDB Database System

Apache CouchDB, the database commonly referred to CouchDB is a document-oriented NoSQL database that is open source. Integration with this system is required for retrieval of the majority of the information that the system will require. Performance for this channel is also crucial since we need fast and superior access speeds since the system mostly relies on querying with the database and thus this system has implemented MVCC - Multi Version Concurrency Control so it can process multiple queries with good response time. Furthermore this database makes use of JSON - JavaScript Object Notation, which transfers encrypted objects and is more dense and more faster than XML - Extensible Markup Language

#### Quality Requirements for the Database System

- **Performance:** All queries to the CouchDB database should be rapid and efficient to provide the best user experience to the maximum amount of people. This is achieved through the MVCC since it can perform multiple queries at once as well as it makes use of JSON, which on its own is fast.
- **Reliability:** It is important that the system needs to have as some down-time for upgrading and maintaining, thus a reliable connection to the Couchbase server is needed.
- **Scalability:** The system needs to be able to work with a large amount of users concurrently. This is met with the MVCC and thus the Couchbase Server is the best suited.
- **Integrability:** The integration with the CouchDB database should be engineered in a manner which can easily be adjusted to accommodate additional integrations. Since this is a Web based system and CouchDB is built for Web since it uses HTTP protocol and JavaScript querying.

- **Affordability:** Access or queries made to the CouchDB database should be affordable, this is met since CouchDB is open source there is no physical money involved and requests made as often as needed are done without any implications, and does not affect the overall system performance.
- **Flexibility:** The integration with the database should be flexible in the sense that slight changes in the database functionality should not affect the integration with the System. This is met since the database has a merging algorithm and manages conflicts on its own.
- **Auditability:** Any modifications made, user interactions, queries should be announced. This can be done through Identification, time-stamps, digital receipts and this should be linked to the integration channels used. Every integration action must be traceable to a user or system action to provide debugging and easy maintenance.

#### 8.0.2 Web services

Web services to be used namely HTTP, HTTPS and TCP, will facilitate user interaction with the system. and provide security to the system.

**Quality Requirements for the Web Services**

- **Security:** This is a very important requirement since in any web-based application needs to be secure if it involves sessions and logging into the system, it must incorporate user authentication and the storage of personal information as well as a digital certificate to make sure that the connection is secure.
- **Reliability:** For this it should be minimalist downtime, since there will be a lot of users and that need the services of the system so it should still have some interaction when down.
- **Integrability:** Web services need to be integrated with a host of other services like Google or any other future organizations that add more functionality and features to the system, typically notification system where the user can get notifications via different methods or services.
- **Performance:** This is a major priority to avoid lowering the overall system performance, since there are a vast range of web services available this must be achieved. However the use of a secured connection can tend to have some performance decrease since there is a lot more to process and is a bit slower than HTTP, but the increased security always is better over performace there us no use if the system is fast but has a security flaw.
- **Scalability:** The protocols utilized need to be able to cope with all the concurrent users connected to the system without breaking down or crashing down and perform optimally.

## 8.1 Access channels

The system has various access channels from which users can gain access to the system:

- Computer - All users of the system should be able to access the system through the Internet/browser. Users will can use a modern web browser, such as Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, or Apple Safari to interact with the system. All users will be required to login to the system to make use of the features available.
- Mobile - Users will also have access to system via an Android application, as well as the web application should have support for the mobile devices.

## 8.2 Protocols

### 8.2.1 Database Query Language

JavaScript will be used to query the CouchDB database by making of MapReduce.

### 8.2.2 HTTP - Hypertext Transfer Protocol

Integration with this protocol will occur at a high level and typically be handled by libraries or browser-clients etc. **Used for:**

- Handling all the socket communications between the client and server, this entails all the data transfer.
- Data transfer between the system and database.
- Transfer of log data such as error codes to ensure both servers and clients are aware of the state of data transfers and its results

### 8.2.3 HTTP - Hypertext Transfer Protocol..

**Used for:**

- Handling all the socket communications between the client and server, this entails all the data transfer.

### 8.2.4 TCP - Transmission Control Protocol..

### 8.2.5 SMTP - Simple Mail Transfer Protocol..

## 8.3 API specifications

### 8.3.1 Apache Maven

### 8.3.2 REST -

### 8.3.3 GIT

GIT can be used as a version control API as it will allow an easy means to manage the versions and also solve code conflicts.

# 9 Technologies

The following technologies will be used to develop the Research System:

- HTML (Hypertext Markup Language): A standardized system for tagging text files to achieve font, colour, graphic, and hyperlink effects on World Wide Web pages. This will be the interface for the users of the system.
- AJAX (Asynchronous JavaScript and XML): Ajax is a group of interrelated Web development techniques used on the client-side to create asynchronous Web applications. The use of Ajax will bring dynamic changes to the web page. This will update changes to the system without the user having to refresh the page and will avoid identical information being submitted multiple times by multiple users.
- JavaEE (Java Platform Enterprise Edition): This platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multitiered, scalable, reliable, and secure network applications. JavaEE is fundamental for the Research System in providing the connections for the user to the system.
- JavaScript (Functionality to HTML): An object-oriented computer programming language commonly used to create interactive effects within web browsers. This will enhance the functionality and appearance of the web pages of the Research System.
- PHP (Server Side Scripting): PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages. The Research System will use PHP for collecting data from forms, such as adding a user and adding a publication.
- MySQL (Database Manager): MySQL is a database system used on the web, which runs on the server. This will be the chosen database system, since it is very fast, reliable, and easy to use, as well as being able to compile on a number of platforms.
- Android (Mobile Devices): Android is a mobile operating system, which is designed primarily for touchscreen mobile devices such as smartphones and tablets. An android application will be developed for the Research System, to provide an easy to use interface for the user on mobile devices.
- Apache (Web Server): The Apache HTTP Server is the world's most used web server software. This is the chosen its performance and reliability, as well as being able to support the other technologies that the Research System will be using.
- Linux/Windows/Mac (Operating System): The three operating systems that are most commonly used today. The Research System will have to be able to run in these three Operating Systems.

# 10 Functional Requirements and Application Design

This section discusses the application functionality and service contracts required by the users.

## 10.1 Use Case Prioritization

The Use Case Prioritisation is specified for each use case in the next section.

## 10.2 Use Case/Services Contracts

Use Case Prioritisation and Service Contracts are described below

### 10.2.1 User Login

**Description:** A user is required to log into the system before the user can make use of any functionality.
**Prioritisation:** Critical

**Pre-conditions**

- A user must have a connection to the server.
- A user must be registered as a user by a person with HOD or Admin permission.
- The user must enter the correct information in order for the authentication to be successful.

**Post-conditions**

- The user has specific access to the server on which all data is stored, i.e add/edit authors and search for authors
- The user is able to use all the user functionality provided by the system
- The user may log out of the system when they wish to.

Figure 1: Service Contract: User Login

### 10.2.2 User Registration

**Description:** In order to be able to log into the system and make use of the functionality provided by the system the user must be registered on the system.
**Prioritisation:** Critical

**Pre-conditions**

- The user must be part of the staff of the Computer Science Department of the University of Pretoria.
- The administrator or HOD of the system must have registered the user on the system.
- The user must decide on their login credentials.

**Post-conditions**

- The user's login information is securely stored in the system database.
- The user can log into the system.

Figure 2: Service Contract: User Registration

### 10.2.3 Update User

**Description:** Personal details as well as log in credentials can be changed if and when necessary.
**Prioritisation:** Important

**Pre-conditions**

- The administrator or HOD must be logged into the system.
- The user to be updated must already exist in the system.
- The administrator or HOD must have the new information.

**Post-conditions**

- The user's information is updated on the system's database.

Figure 3: Service Contract: Update User

#### 10.2.4 Remove User

**Description:** Should a user no longer belong to the Department of Computer Science the person should be removed from the system.
**Prioritisation:** Important

**Pre-conditions**

- The person should be a user on the system.
- The person should no longer belong to the staff of the Department of Computer Science.
- The administrator or HOD must be the logged in.

**Post-conditions**

- The person is removed from the system's database.
- The person can no longer gain access to the system.
- The person history still remains in the log file.

Figure 4: Service Contract: Remove User

#### 10.2.5 Create Publication

**Description:** Adding a paper to the system
**Prioritisation:** Critical

**Pre-conditions**

- The user(may be the administrator or HOD) should be logged into the system.
- The user, if not the administrator or HOD, must be a contributor to the paper.
- All authors who contributed to the paper should be available on the system.

**Post-conditions**

- The paper is added to the system

Figure 5: Service Contract: Create Publication

#### 10.2.6 Update Publication

**Description:** Allows a user/administrator/HOD to change a publication's meta-data
**Prioritisation:** Critical

**Pre-conditions**

- The user(may be the administrator or HOD) should be logged into the system.
- The user, if not the administrator or HOD, must be a contributor to the paper.
- The paper must already be in the system.

**Post-conditions**

- The paper's meta-data is updated

Figure 6: Service Contract: Update Publication

#### 10.2.7 View Publication

**Description:** Allows a user/administrator/HOD to view the meta-data of a paper that lies within their permissions
**Prioritisation:** Important

**Pre-conditions**

- The user(may be the administrator or HOD) should be logged into the system.
- The user - if not the administrator, HOD or research leader - must be a contributor to the paper.
- The paper must already be in the system.

**Post-conditions**

- The paper's meta-data is displayed to the user

Figure 7: Service Contract: View Publication

### 10.2.8 Add Publication Type

**Description:** Allows the administrator or HOD to add a publication type
**Prioritisation:** Important

**Pre-conditions**

- The user(must be the administrator or HOD) should be logged into the system.
- The publication type to add must not already be in the system.

**Post-conditions**

- The new publication type is added to the system.

Figure 8: Service Contract: Add Publication

#### 10.2.9 Update Publication Type

**Description:** Allows the administrator or HOD to update a publication type's details
**Prioritisation:** Important

**Pre-conditions**

- The user(must be the administrator or HOD) should be logged into the system.
- The user - if not the administrator, HOD or research leader - must be a contributor to the paper.
- The publication type to be updated must already be in the system.

**Post-conditions**

- The selected publication type will be updated with the new information.

Figure 9: Service Contract: Update Publication

#### 10.2.10 Add Reminder

**Description:** Allows the user to create a reminder for the deadline of the publication.
**Prioritisation:** Nice-To-Have.

**Pre-conditions**

- The user(can be the HOD) should be logged into the system.
- The user must be a contributor to the paper.
- The publication must already be in the system.

**Post-conditions**

- A reminder via Mail or Calendar will be set with the selected publication and deadline.

Figure 10: Service Contract: Add Reminder

#### 10.2.11 Create Research Group

**Description:** Allows the administrator or HOD to create a new Research Group
**Prioritisation:** Critical

**Pre-conditions**

- The user(must be the administrator or HOD) should be logged into the system.
- The Research Group you wish to create must not already be in the system.

**Post-conditions**

- The Research Group will be created and stored in the database.
- Users will be able to join the Research Group.
- Research Leader will be able to see all Publications made by the Research Group's members.

Figure 11: Service Contract: Create Research Group

#### 10.2.12 View Research Group

**Description:** View the details of a Research Group
**Prioritisation:** Nice-To-Have

**Pre-conditions**

- The user(must be the administrator or HOD) should be logged into the system.
- The Research Group must already exist in the database.

**Post-conditions**

- The user will be able to view the details of the Research Group.

Figure 12: Service Contract: View Research Group

### 10.2.13 Update Research Group

**Description:** Allows the administrator or HOD to update an existing Research Group
**Prioritisation:** Important

**Pre-conditions**

- The user(must be the administrator or HOD) should be logged into the system.
- The Research Group must already exist in the database.

**Post-conditions**

- The user will be able to edit and save the new information entered.

Figure 13: Service Contract: Update Research Group

#### 10.2.14 Remove Research Group

**Description:** Allows the administrator or HOD to remove an existing Research Group
**Prioritisation:** Important

**Pre-conditions**

- The user(must be the administrator or HOD) should be logged into the system.
- The Research Group must already exist in the database.
- The Research Group must have no members in it and no Research Leader.

**Post-conditions**

- The user will be able to remove the Research Group from the database.

Figure 14: Service Contract: Remove Research Group

#### 10.2.15 Create Author

**Description:** Allows the user to create a new Author
**Prioritisation:** Critical

**Pre-conditions**

- The user should be logged into the system.
- The Author must not already exist in the database.

**Post-conditions**

- The user will be able to create a new Author and save him/her in the database.

Figure 15: Service Contract: Create Author

#### 10.2.16 Update Author

**Description:** Allows the user to update an existing Author
**Prioritisation:** Important

**Pre-conditions**

- The user should be logged into the system.
- The Author must already exist in the database.

**Post-conditions**

- The user will be able to edit and save the new information entered.

Figure 16: Service Contract: Update Author

#### 10.2.17 Generate Report

**Description:** Allows the user generate a report based on certain data
**Prioritisation:** Nice to have

**Pre-conditions**

- The user should be logged into the system.
- There should be existing data in the system's database

**Post-conditions**

- The system will generate a report based on certain criteria and make it visible to the user

Figure 17: Service Contract: Generate Report

## 10.3 Required Functionality

### 10.3.1 User-Research System interaction

**Description: The type of user indicates what privileges that user has in the Research system**

**Normal-user**

- A normal user may log in to the system if registered on the system.
- A normal user may add publications to the system.
- A normal user must be an author of a publication should they want to add it to the system.
- A normal user may add authors to a publication.
- A normal user may change authors in a publication.
- A normal user may add a publication to a conference.
- A normal user may only view their own publications.

**Head of Department**

- The head of department may log in to the system.
- The head of department may add/edit/remove a user.
- The head of department may add/edit/remove an author.
- The head of department may add publication/edit a publication.
- The head of department may be an author on a publication.
- The head of department may add authors to a publication.
- The head of department may add/remove publications for conferences.
- The head of department may view all publications on the system.

**Admin**

- Admin users may log in to the system.
- Admin users may add/remove/edit users.
- Admin users may add/edit publications.
- Admin users may not be an author to any publication on the system.
- Admin users may add authors to a publication.
- Admin users may change authors to a publication.
- Admin users may add/remove publications to conferences.

- Admin users may view all publications on the system.

Figure 18: Functional Requirements: Overview of Research System

Figure 19: Functional Requirements: Normal user access privileges

Figure 20: Functional Requirements: Superuser(HOD and admin) access privileges

## 10.4 Process Specification

This section conatins UML activity diagrams that illustrate the sequences that will be followed for various use case scenarios.

Figure 21: Process Specification: Adding A User

Figure 22: Process Specification: Removing A User

Figure 23: Process Specification: Update A User

Figure 24: Process Specification: Adding A Publication

Figure 25: Process Specification: Update A Publication

Figure 26: Process Specification: View A Publication

## 10.5 Domain Model

Figure 27: Domain Model of Research System

# 11 Open Issues

This section deals with issues that still need to be clarified, specified, assumed or have been discovered to include inconsistencies in the requirements, comprising of the following issues:

- Will a server be provided?
- Should the HOD be a separate entity or fall within the Admin entity?
- To where does the publication go to be reviewed?
- Should the reminder system be via Mail or Calendar notification?
- It is assumed that the Venue will have a deadline attached.