
UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

# Software Requirements Specification and Technology Neutral Process Design

*Members:*
Matthew Botha 14214742
Gershom Maluleke 13229908
Nathan Dunkley 14145759
Aiden Malan
Sello Thosago 13062060
Nsovo Baloyi 12163262
Matheu Botha 14284104

## Team India

March 11, 2016
v0.1

# 1 Introduction

The purpose of this document is to outline the core functionality of the new computer science publication system in a technology neutral design specification. This design is not necessarily a depiction of what the final system will look like. During the development of the system it is possible that new functionalities are added, or old ones removed. This is merely the first iteration of the development cycle. Which can be revised in the future. As well as iterativly enhanced and detailed as the need arises.

# 2 Vision

This projects aim is to provide a system to facilitate the administration of postgraduate publications, including but not limited to journal articles and conference papers. It should be able to store information on Users, Authors and Publications as well as employ a authentication system to ensure only specified parties within the system have access to operations on the stored information. The purpose of keeping track of the publications is, in part, to keep track of the DoE and UP weighted units earned by Users and/or authors. The reason for this is that these users/authors are required to publish a certain amount of units each year in order to determine pay, research funding, and maintaining their research position.

# 3 Background

Currently the publications are kept track of with a complicated excel spreadsheet. The purpose of the spreadsheet is to keep track of the status of publications in order to generate a report detailing the units earned by a researcher. There are many drawbacks to this system. It is tedious to maintain the spreedsheet. Data is spread over multiple sheets. The larger the database gets the slower it preforms. The information is displayed in a raw textual format which makes it a daunting task to sit through and analyse. It is difficult to add new functionality and change or enhance existing functionality. When pulling in data from multiple sheets it becomes difficult to debug the Lookups.

# 4 Software Architecture Documentation

## 4.1 Architecture requirements

### 4.1.1 Architectural scope

Persistence infrastructure is important as storing data is one of the core functionalities of a functional system. There are many options with regard to maintaining data persistence

- MySQL - This is a safe choice as most programmers are familiar with MySQL. This allows fast, reliable development.Its An Industry Standard (And Still Extremely Popular). It is also compatible with most operating systems which makes deployment easy. There are also many well maintained libraries that can be used in most programming languages.

- MongoDB - This is becoming increasingly popular. It is a no sql database that stores the data in a similar format to json objects. This makes the database sizes more manageable and also allows operations such as database backups and migration to be performed far easier.

With regards to the reporting infrastructure we need to consider the need to generate the reports in a downloadable fashion.

- CSV - Comma Seperated Value. These files are the most common and most basic way to export a set of data in a organised table format. This is most likely the approach that will be taken for the exportation of the variable data

We also need to consider the fact that report also have graphs. Thus we need a way to export the graphs, both to the interfaces and to the user.

- SVG - Scalable Vector Graphic. This is widely prefered format type for data representation images as it allows for arbitrary levels of zoom. This is due to the fact that the images are defined as a set of verticies so a computer can display these verticies to any degree of granularity.

- JPEG - This is another prefered image format type as the images themselves are im a compressed format and the format is almost universaly supported.

#### 4.1.2 Quality requirements

- Reliability - The reports generated from the data provided to the system should be reliable.

- Scalability - It is estimated that there will be approximatly one hundred users. The system should be able to handle each user being active at the same time even though it is unlikly.

- Cost - Generating a report for all data in the system will be the worst case scenario in this system and thus have a high cost. The system should however be able to produce subsets of the report at a relativly low cost. For example a report for all researchers under a specific research group.

- Security - There should be strict access right applied to each of the subsystems and their components. This means that if a user, by definition of the buisness rules, is not allowed to access a certain functionality, then they will be denied access.

- Auditability - Actions performed on the system should be logged in a manner that would allow one to trace back the history of actions performed on the system.

#### 4.1.3 Integration and access channel requirements

The system needs to be accessed by a web interface as well as an android interface. For the web integration it will be done using the REST architecture. In the REST architecture ecerything is treated as a resource. A restful service is usually highly scalable and maintainable. Resources are identified by URIs. The URI specifies an adress for the requested resource in a global adress space. Integration with the android interface will be similar as android apps communicate with http requests.

#### 4.1.4 Architectural constraints

Client has not provided any architecture constraints.

### 4.2 Architectural patterns or styles

On this section we will be describing the different patterns and styles to be used when developing the CS Research System.

#### 4.2.1 Client-server pattern

- Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
- The main reason why we chose this pattern is because sensitive and personal data pertaining to the name, emails and other client information must be kept secure. Client/Server architecture provides a far higher degree of security and data integrity. The nature of the CS Report System necessitates a means of centralised data access and a system featuring high maintainability. The Client/Server architecture ensures that changes made to the system are immediately visible to all clients.

##### 4.2.1.1 Server

- The Server handles storage of data into the database and provides data on request. The Server is responsible for all back end functions of the software including the verification of users for security.

##### 4.2.1.2 Client

- The client is responsible for the front end of the CS report system. It also allows users to send queries to the server and requesting for various

services offered by the server. The client also provides an interface between the system and its users.

#### 4.2.2 Authentication Enforcer pattern

- The Authentication Enforcer pattern handles the authentication logic across all of the actions within the Web tier. It assumes responsibility for authentication and verication of user identity and delegates direct interaction with the security provider to a helper class. This applies not only to password-based authentication, but also to client certicate-based authentication and other authentication schemes that provide a users identity, such as Kerberos.

- This pattern will help us improve security of the system while keeping the scalability and performance well maintained since the Authentication Enforcer pattern provides a consistent and structured way to handle authentication and verication of requests across actions within Web-tier components and also supports MVC architecture without duplicating the code.

#### 4.2.3 Dependency Injection

- Dependency injection implements inversion of control for software libraries. The design pattern allows a client to remove all knowledge of a concrete implementation that it needs to use. This helps isolate the client from the impact of design changes and defects. It promotes re-usability, testability and maintainability

- The benefits of using Dependency Injection in our system, is that it will ensure loose coupling of code between the dierent modules that need to be implemented. Decoupling of code will ensure that the code in the system are cleaner, easier to modify when required and easier to adapt and implement for reuse.

#### 4.2.4 Blackboard pattern

- The blackboard pattern allows multiple processes to work closer together on separate threads, polling and reacting if needed. The CS Report System will be separated through layers. There will be User Interface layer, services layer and process layer which will include Business logic and data. User Interface layer will handle interaction like receiving input like login credentials from users. The service layer will handle services like opening a research group and deleting a research group and lastly process layer will process services rendered for authorization. Separation through layers will enhance performance, how the software reacts, manageability and reusability.

#### 4.2.5 MVC

- Model View Controller divides the web portion of the server into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. This ensures reusability of code, easy to maintain code and an added maintenance benet as well as ensure separation of concerns, cohesion, decoupling of components and pluggability on the CS Report System.

##### 4.2.5.1 Model

- The Model directly manages manages the data on the database and the overall flow of the software.

##### 4.2.5.2 View

- Data can be represented and structured sufficiently using HTML.

##### 4.2.5.3 Controller

- The controller will accept the input from the user and convert it into commands that the model or view can understand.

### 4.3 Architectural tactics or strategies

#### 4.3.1 Reliability

Any data that is added to the system must be checked before it is stored. Any incorrect data (e.g. letters in a date) must be fixed before the form can be submitted. This ensures that the stored information is consistent.

A strict format must be applied to any reporting and any generated report must follow the format. If needed data is missing, there should be stand-in messages (such as "No data available").

#### 4.3.2 Scalability

In order to reduce the load of multiple users, use cases that are common (such as some reports) should be optimized.

If multiple users have access to the same resources, the addition of data should be done on a first-come-first-serve basis, and the newest information should override older data.

When the amount of users logged in exceeds the load capabilities of the server, the server should be able to scale up in order to handel the excess requests.

#### 4.3.3 Cost

Reports could be pre-processed and cached so that the same report is not generated multiple times, saving on resources. The system should be able to produce subsets of the report at a relativly low cost (e.g. a report for all researchers under a specific research group), and re-use those subsets to make the larger reports.

#### 4.3.4 Security

Strict user control must be enforced, ensuring that each step of any use case makes sure that the user has the correct authorisation. By using hashing, passwords can be stored safely in the database, with a lower risk to user information leaks. **No plaintext passwords should be stored**.

#### 4.3.5 Audibility

By using multiple log files, the different kinds of actions can be individually logged (i.e. individual logs for Errors, Logins, Data changes etc.), and there should be log files for individual days (a new log for each day) so that the information is organised. Over time, old log files can be merged together for archiving purposes.

### 4.4 Use of reference architectures and frameworks

The system will incorporate the Ruby on Rails framework. It is easy to use this framework with front end frameworks. Ruby on Rails will be coupled with jQuery. The system will use jQuery because it is extensible and supported by most, if not all, browsers. Node.js will be used because it is a runtime environment which runs and handles all the server-side logic.

The system will make use of the Model-view-controller architecture. This architecture is ideal because the software application will be separated into three interconnected parts, which means each module can function as separate entities.

### 4.5 Access and integration channels

#### 4.5.1 Access Channels

The user will be able to access the system, provided they are logged in, via the web either with a PC or a smartphone. It is important that the system has a user friendly interface, especially for mobile devices, to make it as easy to use as possible. The system's access channels will involve HTML5, JavaScript for client side scripting, CSS for styling and Bootstrap for a responsive design. The user will also be able to use the system via an Android application if they so choose. This will give the user the best experience when interfacing with the system via a smartphone.

### 4.5.2 Integration Channels

The system will have to integrate the mobile app with the database and the website to ensure that everything works concurrently and correctly. Integration channels of the website will be html, php, javascript and css. These form part of the basic tools to intergrate any website together. NodeJS will be used as a platform on which the mobile app will be created. The system will integrate with external systems via a MongoDB database. NodeJS and MongoDB will have to be integrated together so that the mobile app works correctly and efficiently. Finally all systems as a whole will have to use the same central database and this is how the system is linked together.

## 4.6 Technologies

When designing a system the consideration of technologies or platforms, that will be used, is of great importance. This section will describe the technologies that will be used for the CS report system, as well as give reasons for the choices made. The CS report system will bring together several technologies to make one working system. Since the system will include two facets, the web interface and the mobile interface, the choice of technologies is important. Regardless of the technologies chosen the system must remain must remain efficient and functional on both interfaces. The technologies that will be used include and not limited to:

**Database:**

- **MongoDB** is a cross platform document-oriented database. This database, because it is a NoSQL database. NoSQL over the traditional SQL relational database for its speed when accessing data. MongoDB also stores its data in a similar format to JSON which will go well with the other technologies to be mentioned later in the text.

**Web Interface Technologies:**

- **HTML5** is latest version of the standard markup language for displaying information/ webpages using the HTTP or HTTPS protocols on the web.
- **CSS3 together with the Bootstrap framework** for the styling of the webpages. CSS is the standard style sheet language used with HTML. The Bootstrap framework provides preprogramed styles that can be used to our convenience.
- **JavaScript**, in particular the AngularJS framework for providing functionality in the webpage. JavaScript is a client-side language and will provide a way for the webpage to communicate with the database and server. AngularJS over jQuery because it provides more functionality and works well with NodeJS and MongoDB.
- **NodeJS** for the server-side scripting. It will communicate with the database. It is fast and works well with AngularJS and MongoDB.

**Mobile Interface Technologies:**

- **Java and XML** will be used for the Android application development. Java and XML are used together to make Android applications.
- **Gradle** is a build system. It is the default build system in Android Studio

**Operating Systems:**

- **PC Platforms**

  The system will run on the 3 most widely used platforms namely Linux, MacOS and Windows. It will be supported by Google Chrome, Safari and Mozilla Firefox. The server will run on a Linux platform.

- **Mobile Platforms**

  Android OS will be the focal mobile OS because of its popularity; haven said that, the system can be adapted to run on iOS and Windows Mobile as well.

**Other Technologies to be considered:**

- **Google Calendar API**. The Google calendar API can be integrated into both the web based and mobile interfaces. This will allow users to enter their schedule e.g. Due date for reports, conference dates

# 5 Functional Requirements and Application Design

## 5.1 Use case Prioritisation

### 5.1.1 User Subsystem

Register User: Critical
Update User: Important
Login User: Critical
Logout User: Critical
Add to Publication: Important

### 5.1.2 Publication Subsystem

**Create Publication:** Important
**View Publication:** Nice-to-Have
**Update Publication:** Important
**Remove Publication:** Important
**Add Author:** Important
**Remove Author:** Important

#### 5.1.3 Conference Subsystem

**Add Conference:** important
**Update Conference:** important
**Delete Conference:** important

#### 5.1.4 Journal Subsystem

**Add Journal:** important
**Update Journal:** important
**Delete Journal:** important

#### 5.1.5 Thesis Subsystem

**Add Thesis:** important
**Update Thesis:** important
**Delete Thesis:** important

#### 5.1.6 Research and Authors Subsystem

**Create research paper:** important
**Update research paper:** important
**Remove research paper:** important
**View research paper:** nice to have

#### 5.1.7 Report Subsystem

**Log In to System:** critical
**View Available Reports:** important
**Select Filters to Apply to Reports:** important
**Select Output Type of Report:** nice-to-have
**Show Report:** important

### 5.2 Use case/Services contracts

#### 5.2.1 User Subsystem

##### 5.2.1.1 Pre-conditions:

**5.2.1.2 Register user:** The user can not be registered to the system already and the user registering a new user has to be admin.

**5.2.1.3 Update user:** The user has to be logged in first before they can update their profile and they can only update their own profile.

**5.2.1.4 Login user:** The user has to be registered to the system first.

**5.2.1.5 Logout user:** The user has to be logged in first.

**5.2.1.6 Add to publication:** The user has to be logged in first and the publication has to exist in the system

**5.2.1.7 Post-conditions:**

**5.2.1.8 Register user:** The user is now registered and added to the database.

**5.2.1.9 Update user:** The user's profile is updated and the changes are saved to the database.

**5.2.1.10 Login user:** The user is logged into the system.

**5.2.1.11 Logout user:** The user is logged out of the system.

**5.2.1.12 Add to publication:** The user's name is now added to the author's of the selected publication.

### 5.2.2 Publication Subsystem

**Use Cases:**

**3.1 Create Publication:**

**Pre-conditions:**The user must be logged into the system

**Post-conditions:**The publication is created

**3.2 View Publication:**

**Pre-conditions:**The user must be logged onto the system and the publication must be created before hand

**Post-conditions:**The user will be able to view and edit(Add authors etc.) to the publication.

**3.3 Update Publication:**

**Pre-conditions:** The user must be logged onto the system and the publication must be created before hand

**Post-conditions:**The publication is updated

**3.4 Remove Publication:**

**Pre-conditions:**The user must be logged onto the system, the publication must be created before hand and the user must confirm the removal of the publication

**Post-conditions:**The publication will be removed

**3.5 Add Author:**

**Pre-conditions:**The user must be logged onto the system and the publication must be created before hand

**Post-conditions:**A new author is added to the publication

**3.6 Remove Author:**

**Pre-conditions:**The user must be logged onto the system, the publication must be created before hand and the user must confirm the removal of the selected author

**Post-conditions:**The author is removed from the publication

#### 5.2.3 Conference Subsystem

**Add Conference:**
**Pre-conditions:** The user must be logged in.
**Post-conditions:** The conference is added.
**Update Conference:**
**Pre-conditions:** The user must be logged in and a the conference must be added prior to this use case.
**Post-conditions:** The conference is updated.
**Delete Conference:**
**Pre-conditions:** The user must be logged in and a the conference must be added prior to this use case.
**Post-conditions:** The conference is deleted.

#### 5.2.4 Journal Subsystem

**Add Journal:**
**Pre-conditions:** The user must be logged in.
**Post-conditions:** The journal is added.
**Update Journal:**
**Pre-conditions:** The user must be logged in and a the journal must be added prior to this use case.
**Post-conditions:** The journal is updated.
**Delete Journal:**
**Pre-conditions:** The user must be logged in and a the journal must be added prior to this use case.
**Post-conditions:** The journal is deleted.

#### 5.2.5 Thesis Subsystem

**Add Thesis:**
**Pre-conditions:** The user must be logged in.
**Post-conditions:** The thesis is added.
**Update Thesis:**
**Pre-conditions:** The user must be logged in and a the thesis must be added prior to this use case.
**Post-conditions:** The thesis is updated.
**Delete Thesis:**

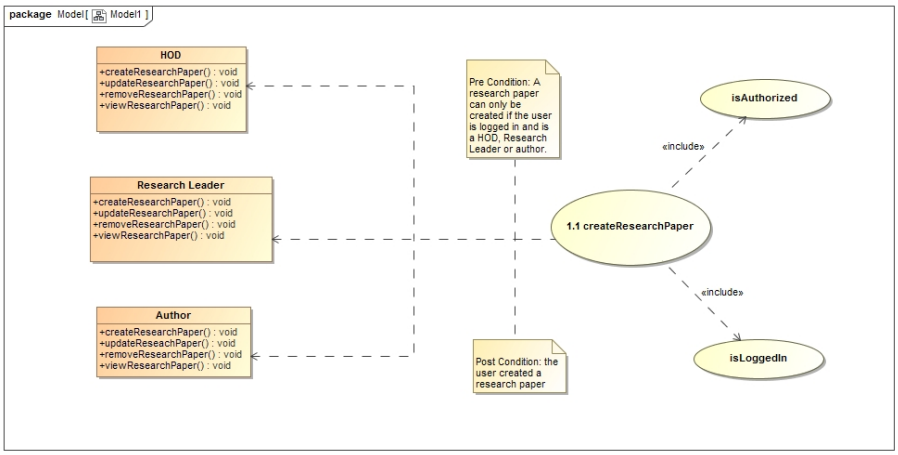



Figure 1: Pre and Post conditions create function

**Pre-conditions:** The user must be logged in and a the thesis must be added prior to this use case.

**Post-conditions:** The thesis is deleted.

#### 5.2.6 Researchers and Authors Subsystem

**3.1 Create Research Paper:**

**Pre-conditions:**The user must be logged into the system. The user must also be either HOD, research leader or author.

**Post-conditions:**The research paper is created.

**3.2 Update Research Paper:**

**Pre-conditions:**The user must be logged onto the system and the research paper must exist.

**Post-conditions:**The user will be able to view and edit(Add authors etc.) to the research paper.

**3.3 Remove Research Paper:**

**Pre-conditions:** The user must be logged onto the system and the research paper must exist.

**Post-conditions:**The the research paper is deleted.

**3.4 View Research Paper:**

**Pre-conditions:**The user must be logged onto the system, the research paper must exist.

**Post-conditions:**The publication can be viewed

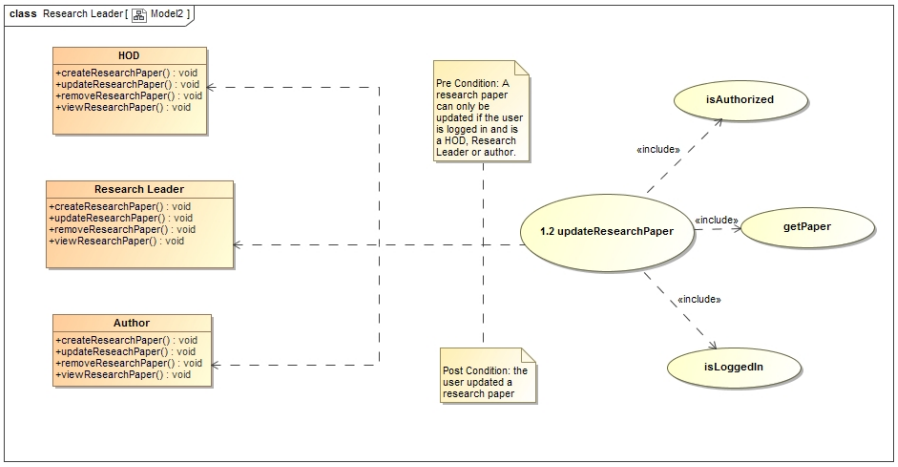



Figure 2: Pre and Post conditions update function

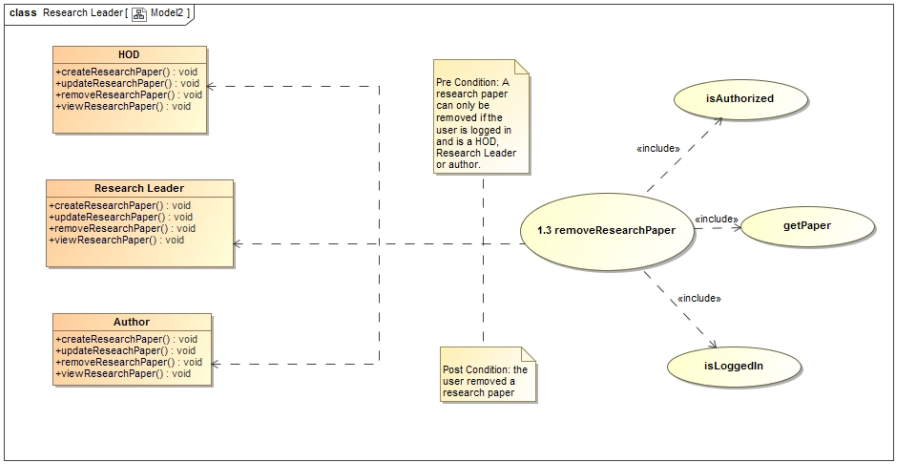



Figure 3: Pre and Post conditions remove function

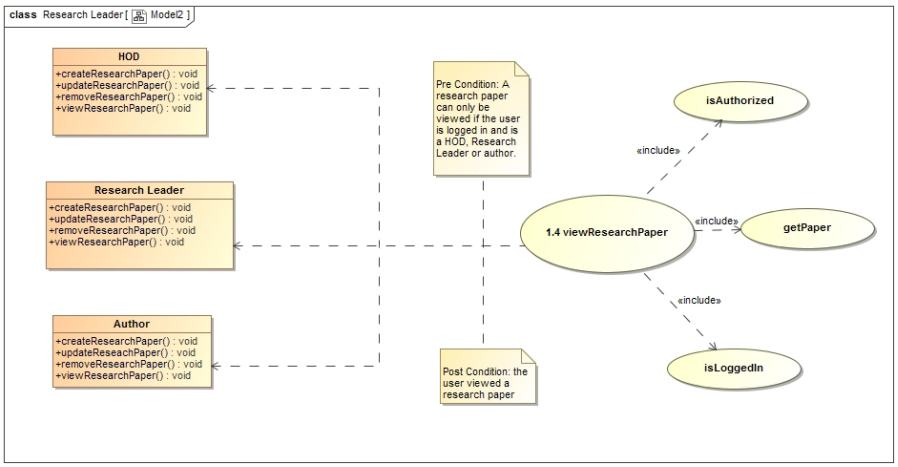



Figure 4: Pre and Post conditions view function

### 5.2.7 Conference, Journal and Thesis Subsystem Request and Results Data Structures

### 5.2.8 Report Subsystem

**Pre-Conditions:** User must be logged in to be able to generate a report.
**Post-Conditions:**

- A simple but comprehensive report must be generated in the format chosen by the user.
- The report must only include papers that the current user is associated with.

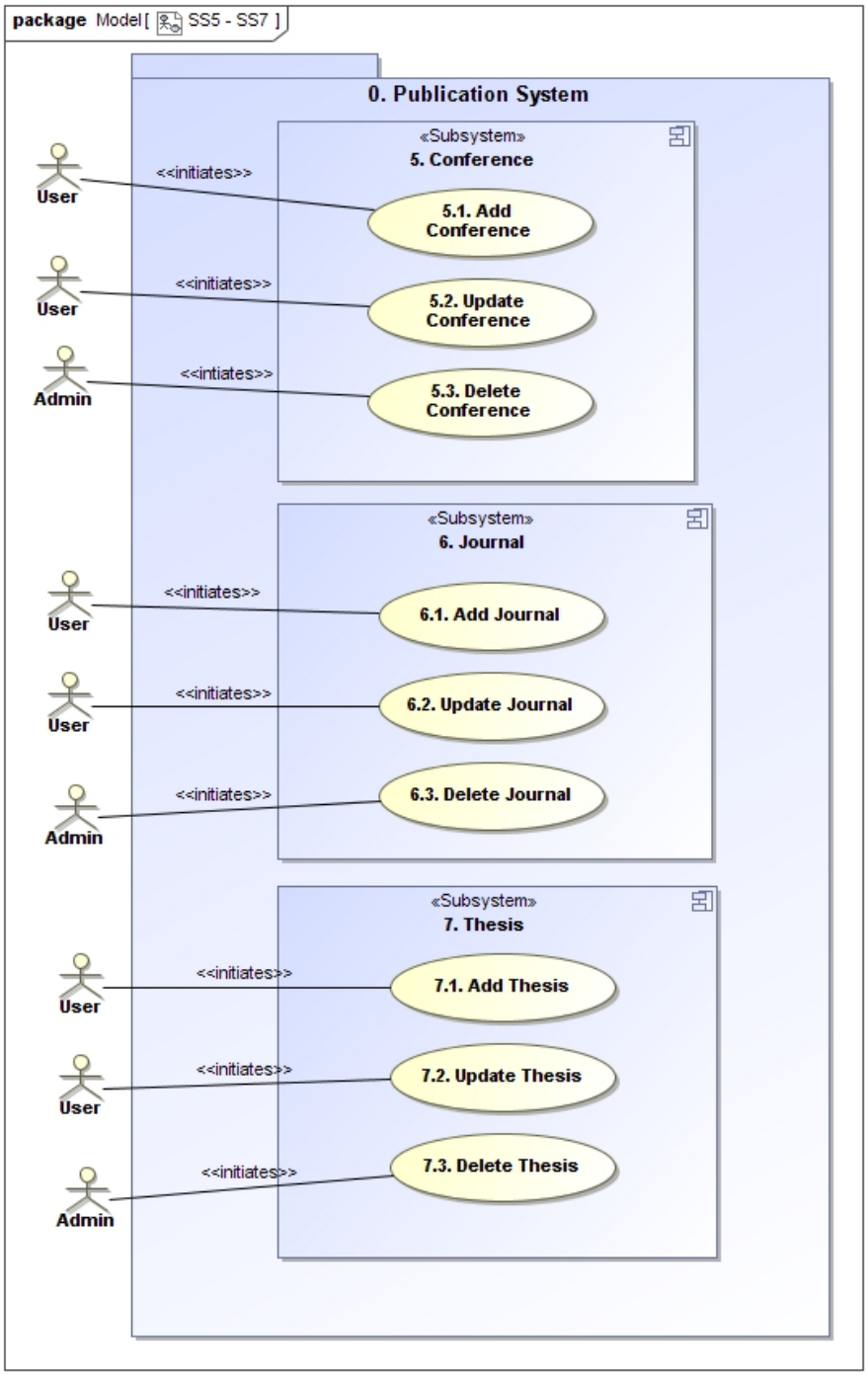
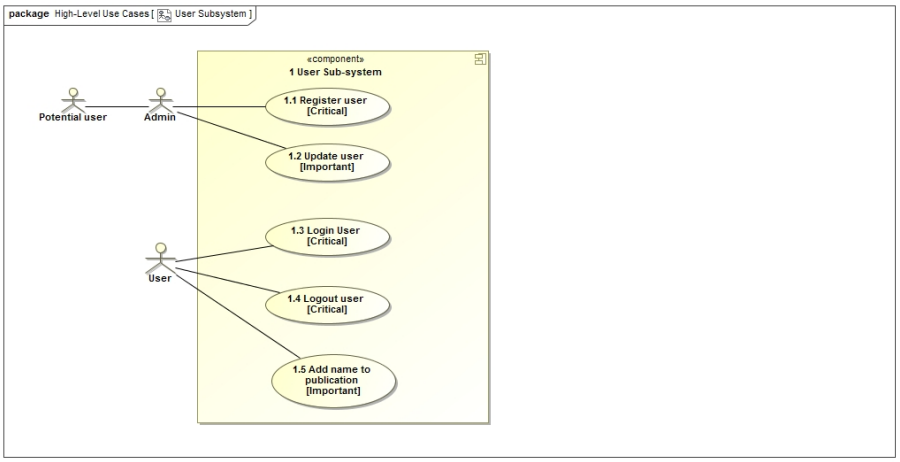
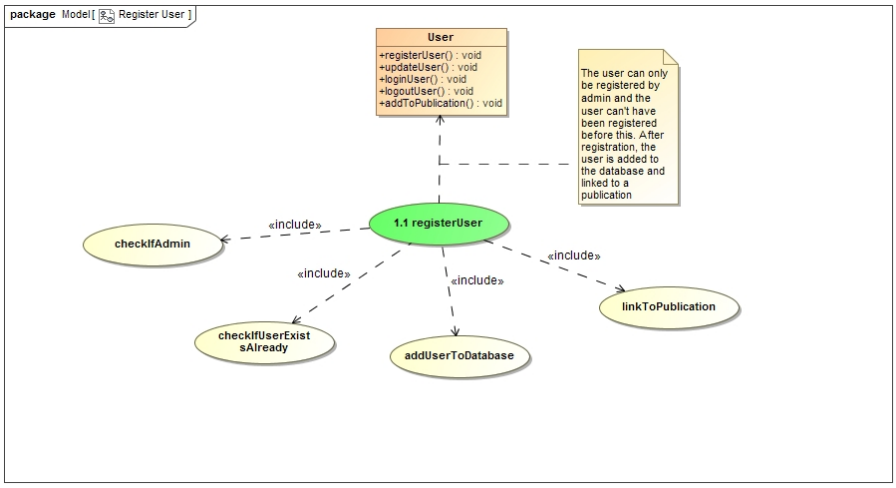


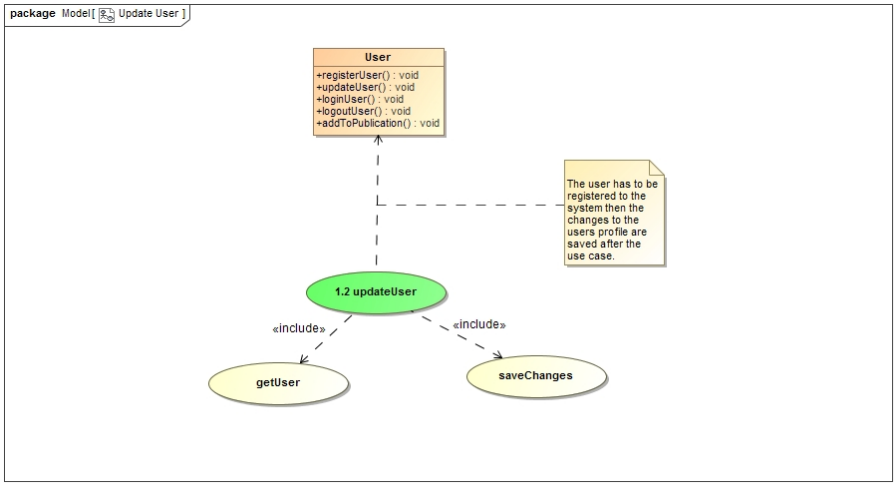

Figure 5: Use case diagram for the Conference, Journal and Thesis Subsystems

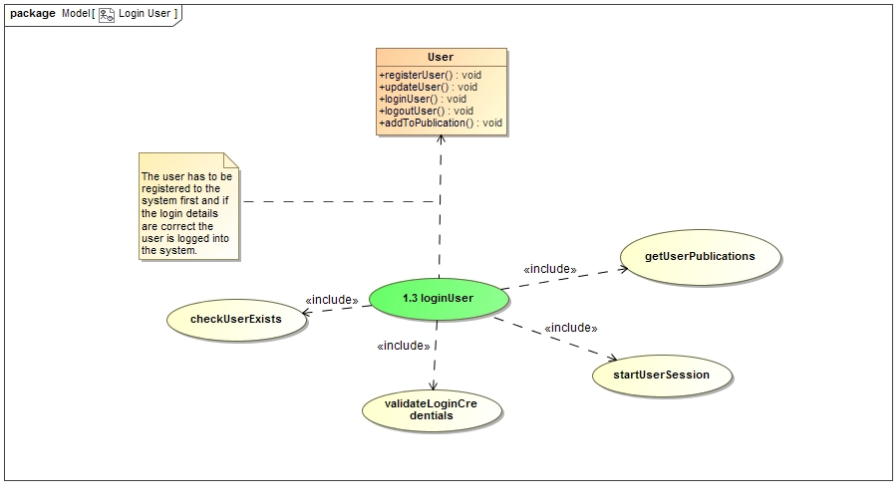
## 5.3 Required Functionality

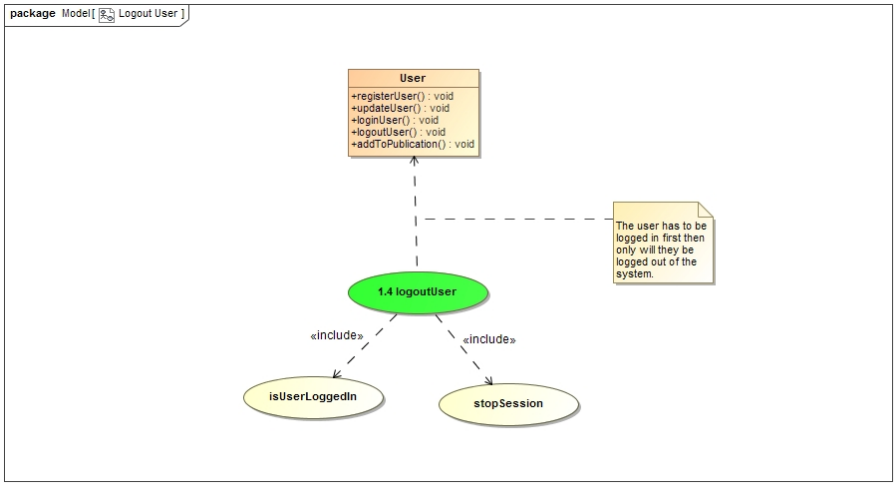
### 5.3.1 User Subsystem Use Case

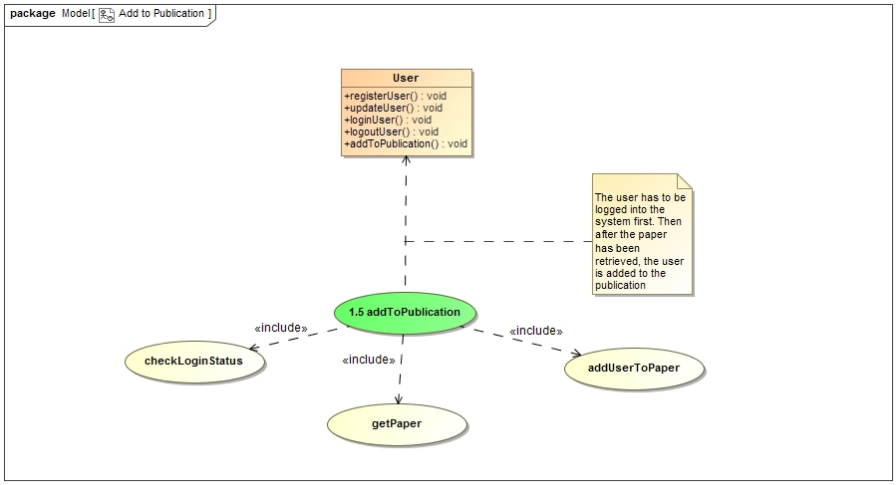



package Model[ Update User ]
User
+registerUser() : void
+updateUser() : void
+loginUser() : void
+logoutUser() : void
+addToPublication() : void
The user has to be registered to the system then the changes to the users profile are saved after the use case.
1.2 updateUser
«include»
«include»
getUser
saveChanges


package Model[ Login User ]
User
+registerUser() : void
+updateUser() : void
+loginUser() : void
+logoutUser() : void
+addToPublication() : void
The user has to be registered to the system first and if the login details are correct the user is logged into the system.
getUserPublications
«include»
«include»
1.3 loginUser
checkUserExists
«include»
«include»
startUserSession
validateLoginCredentials


package Model[ Logout User ]
User
+registerUser() : void
+updateUser() : void
+loginUser() : void
+logoutUser() : void
+addToPublication() : void
The user has to be logged in first then only will they be logged out of the system.
1.4 logoutUser
«include»
«include»
isUserLoggedIn
stopSession


package Model[ Add to Publication ]
User
+registerUser() : void
+updateUser() : void
+loginUser() : void
+logoutUser() : void
+addToPublication() : void
The user has to be logged into the system first. Then after the paper has been retrieved, the user is added to the publication
1.5 addToPublication
«include»
«include»
«include»
checkLoginStatus
addUserToPaper
getPaper

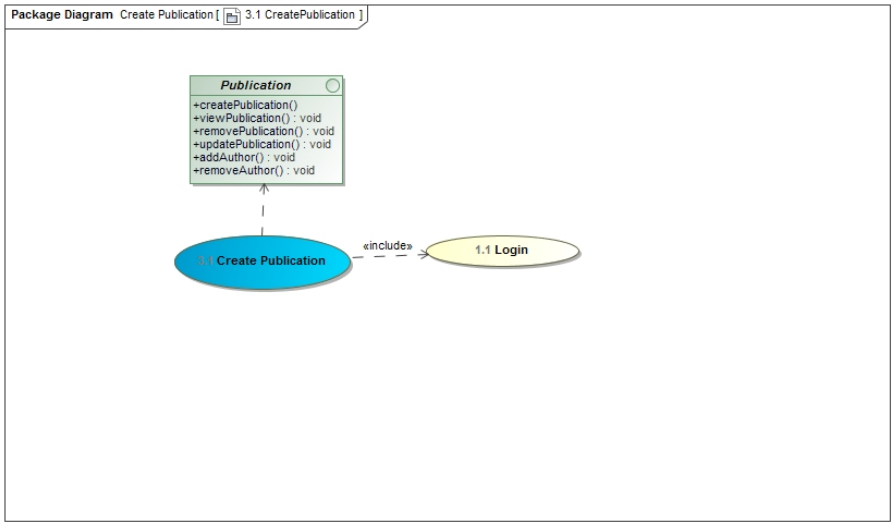
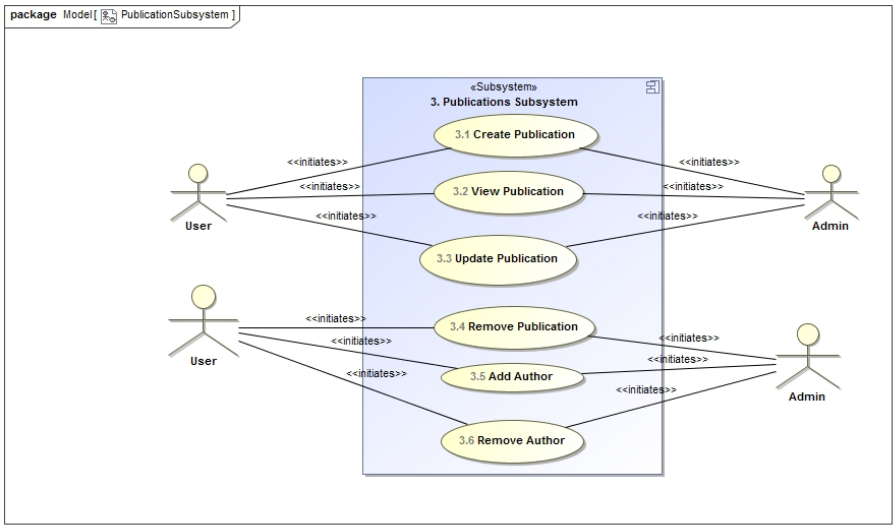



Figure 6: Create Publication

### 5.3.2 Publication Subsystem Use Case



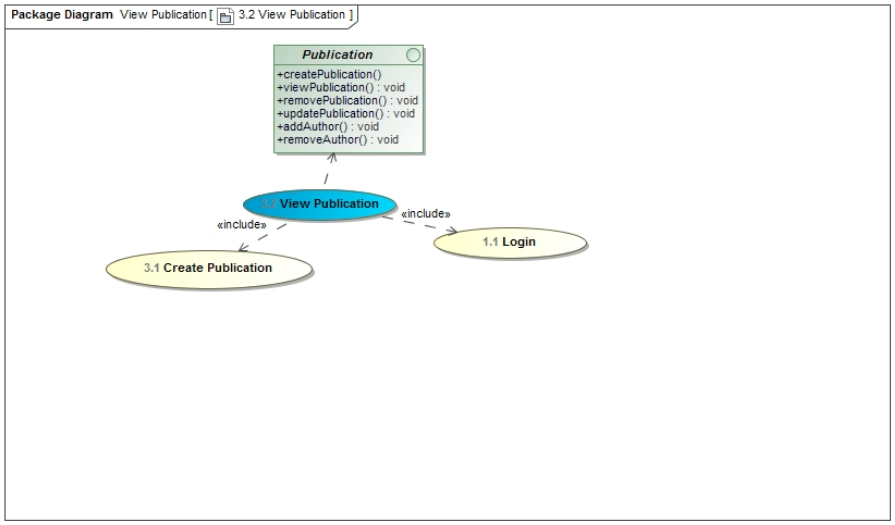



Figure 7: View Publication

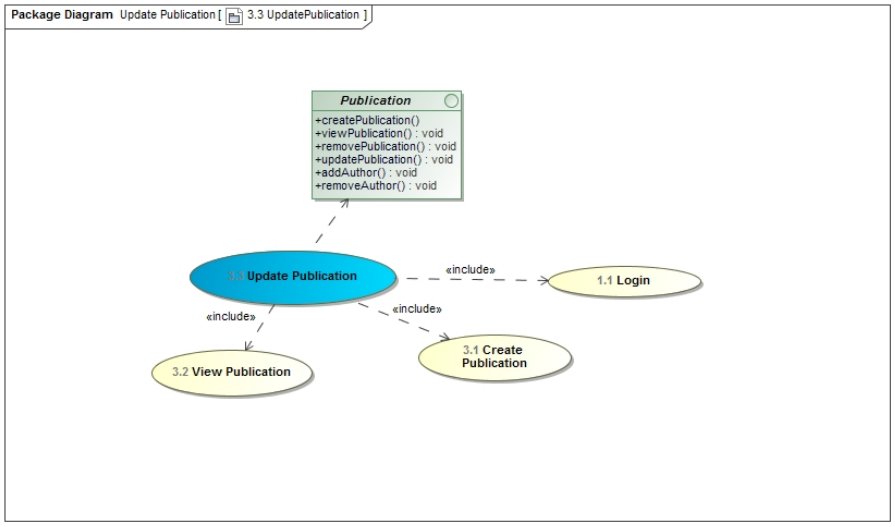



Figure 8: Update Publication

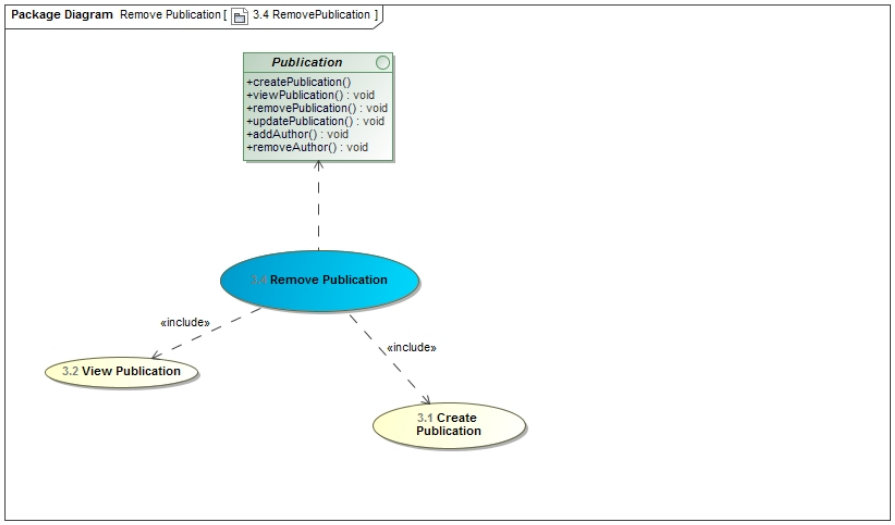



Figure 9: Remove Publication

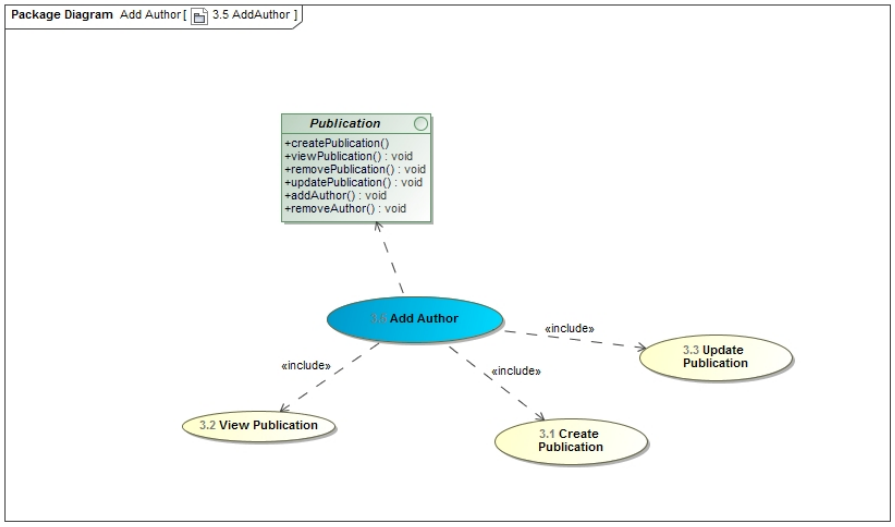



Figure 10: Add Author

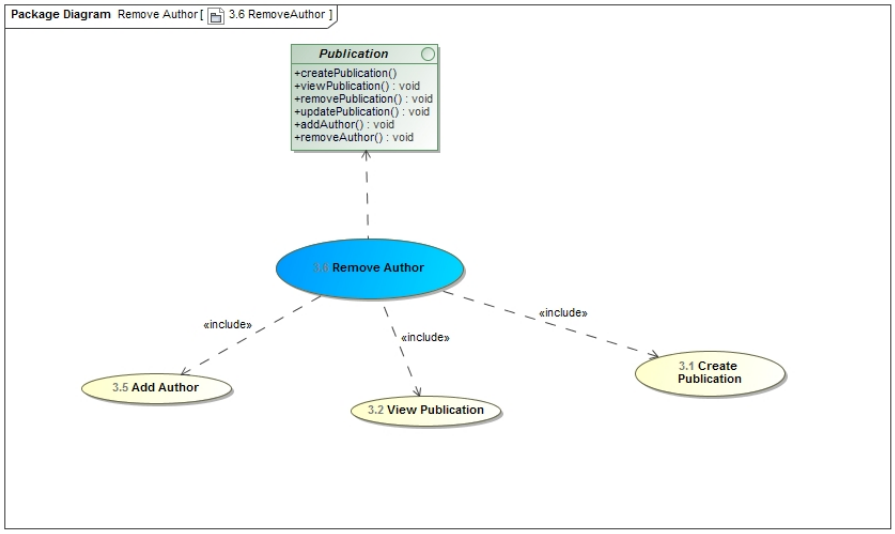
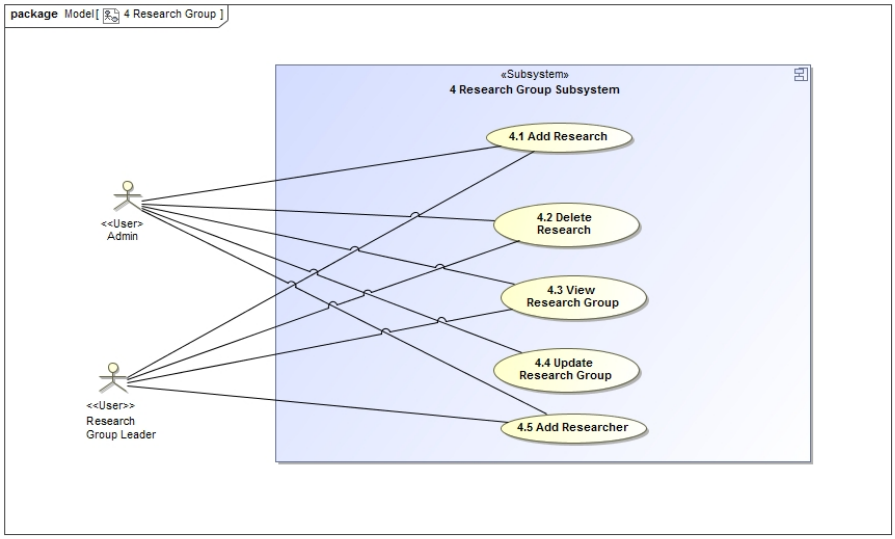



Figure 11: Remove Author

### 5.3.3 Research Group Subsystem Use Case



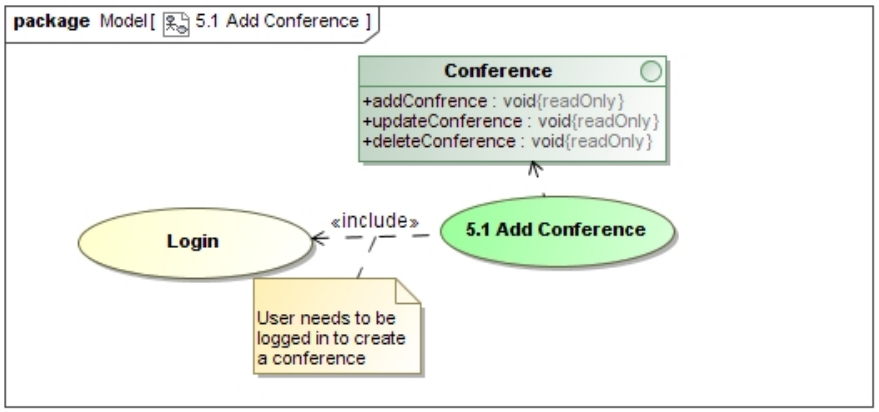



Figure 12: Add Conference

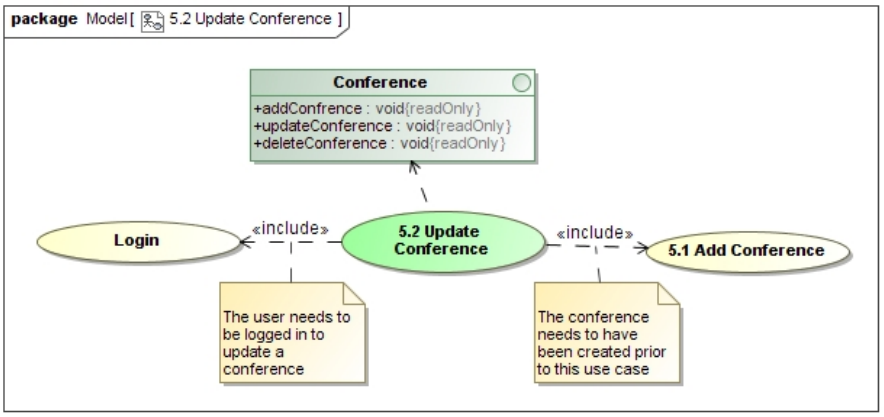



Figure 13: Update Conference

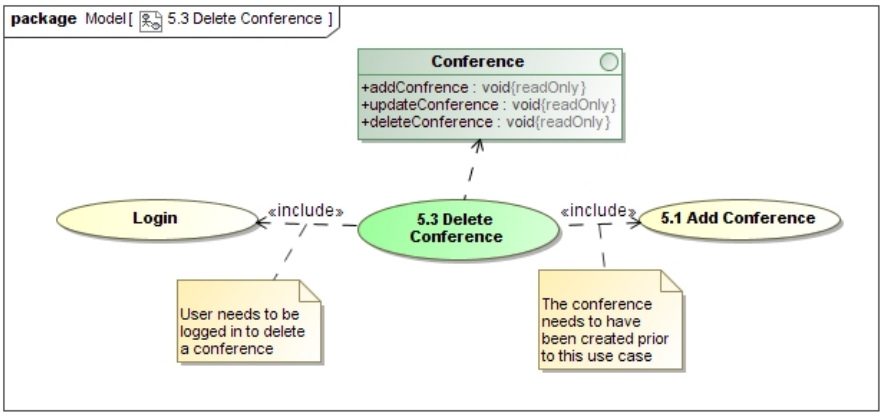



Figure 14: Delete Conference

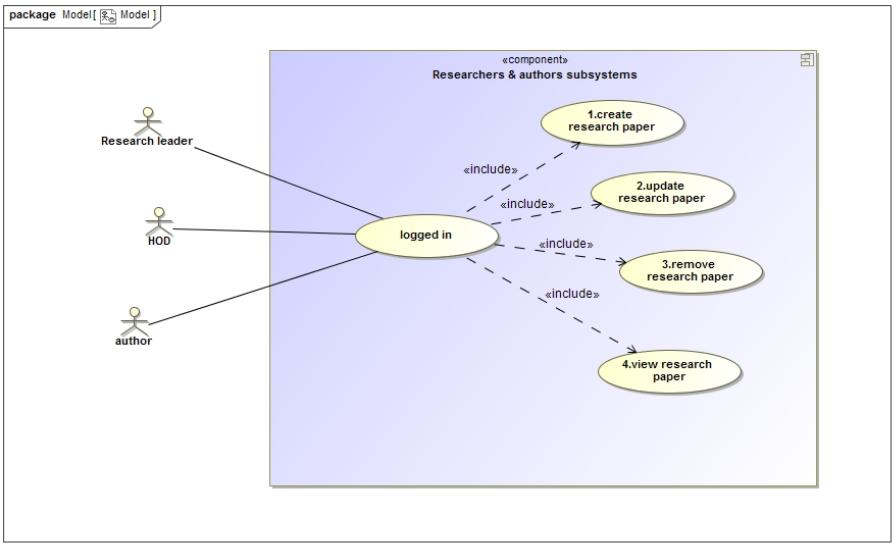



Figure 15: Research and authors use case diagram

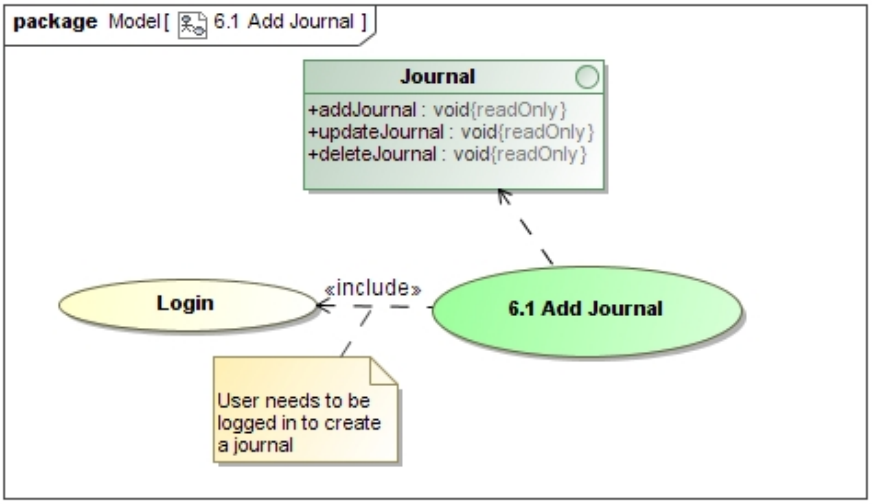



Figure 16: Add Journal

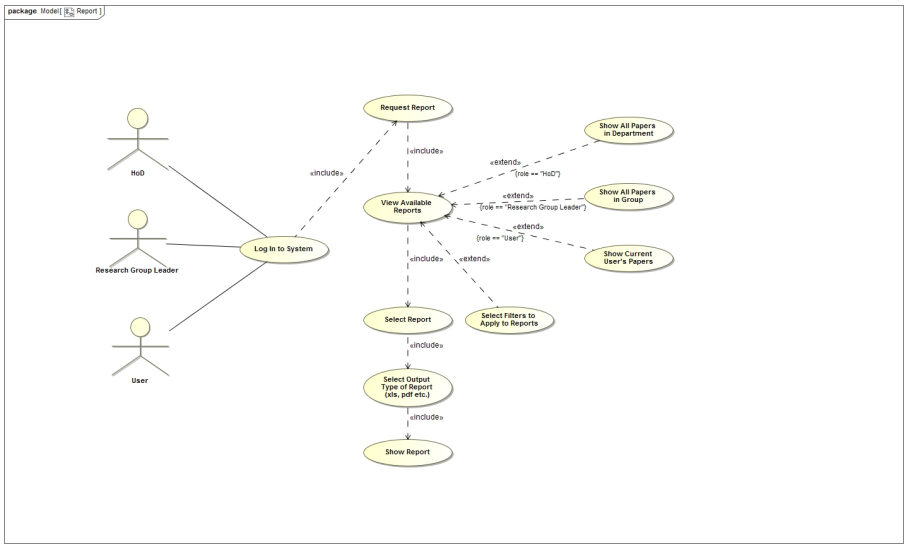
### 5.3.4 Conference Subsystem Use Case

### 5.3.5 Research and authors Subsystem Use Case

### 5.3.6 Journal Subsystem Use Case

### 5.3.7 Thesis Subsystem Use Case

### 5.3.8 Report Subsystem Use Case



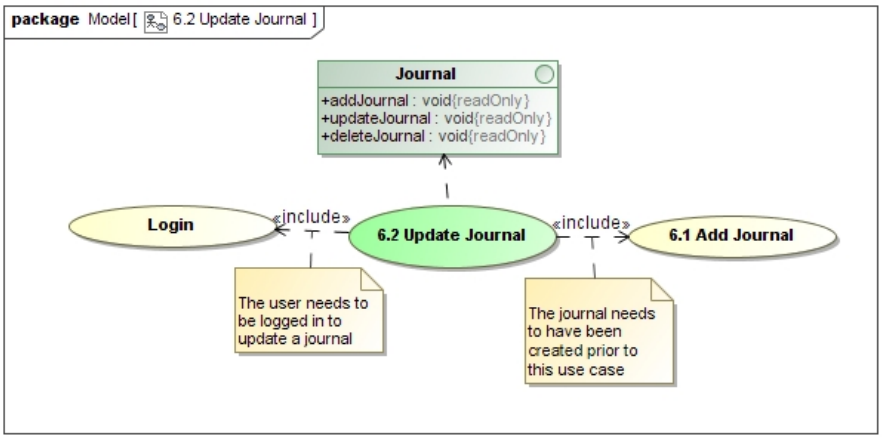



Figure 17: Update Journal

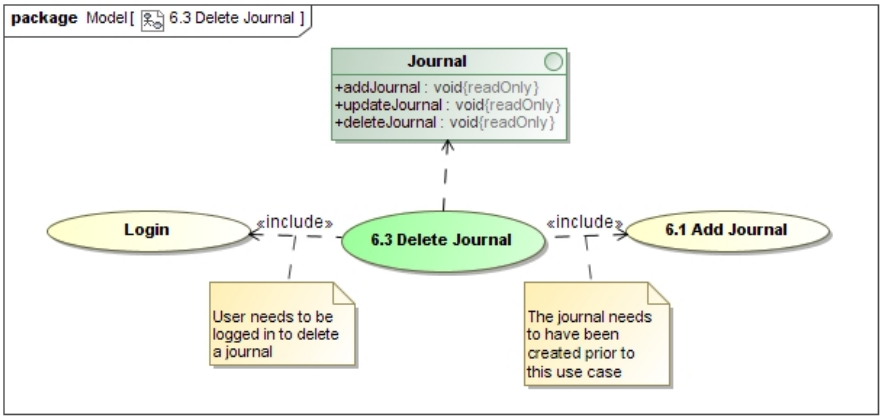



Figure 18: Delete Journal

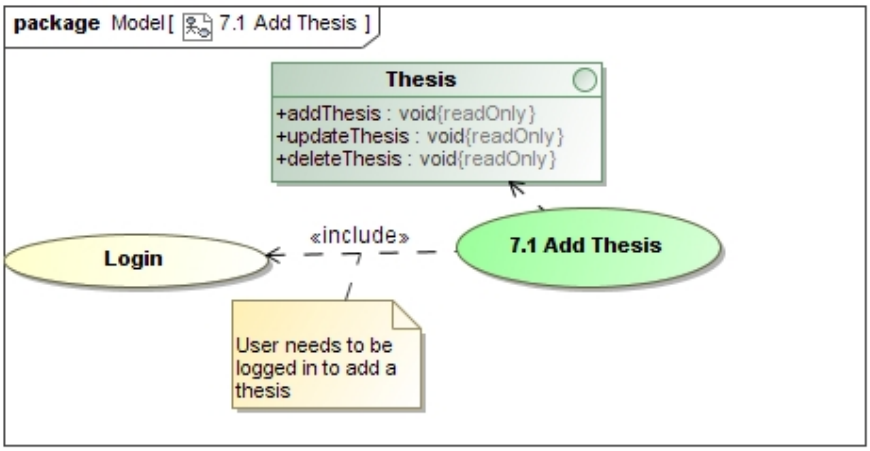



Figure 19: Add Thesis

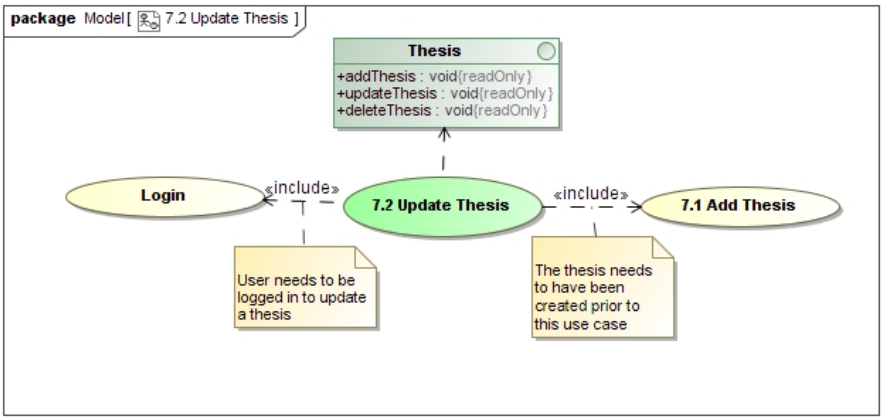



Figure 20: Update Thesis

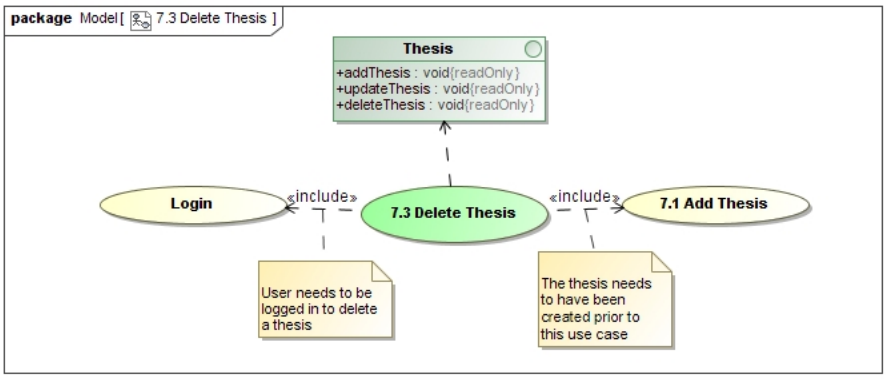



Figure 21: Delete Thesis

## 5.4 Process Specifications

### 5.4.1 Conference Subsystem

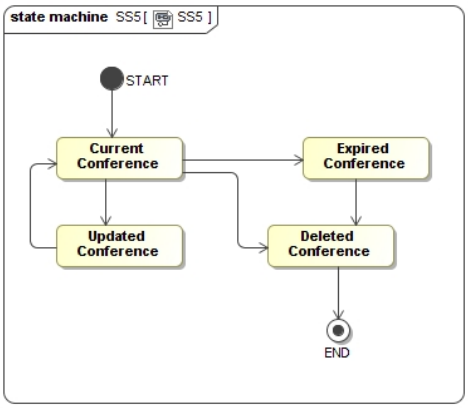



Figure 22: Conference State Diagram

### 5.4.2 Journal Subsystem

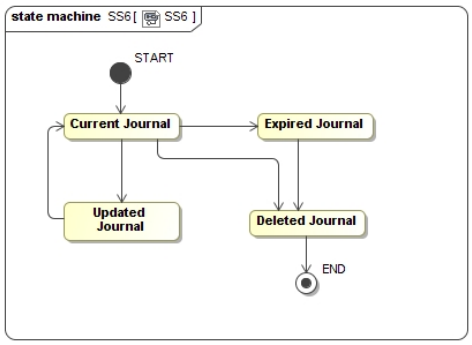



Figure 23: Journal State Diagram

### 5.4.3 Thesis Subsystem

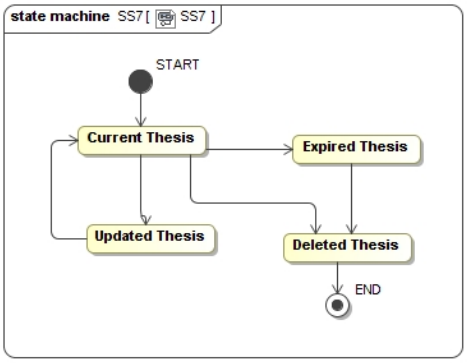



Figure 24: Thesis State Diagram

## 5.5 Domain Model

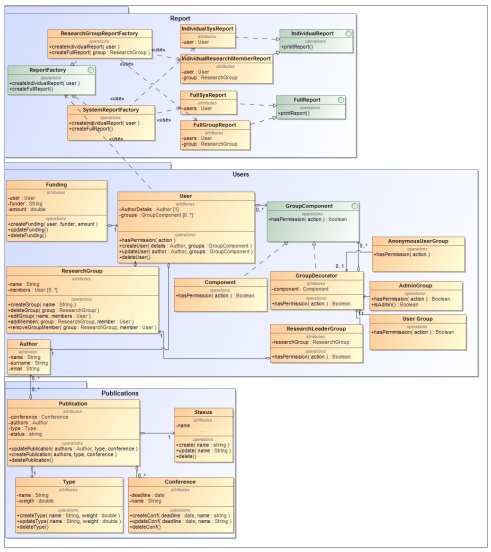



Figure 25: Domain Model

# 6 Open Issues

- Specifics with regard to the DoE and UP weighted contributions
- Further elaboration on how funding occurs.