CodingInfinity

Benchmark Service Functional Requirements Documentation

Git:
`https://github.com/CodingInfinity/Benchmark-Service-Documentation`
GitHub Organization: `https://github.com/CodingInfinity`

The Client:
Ms Vreda Pieterse
Department of Computer Science
University Of Pretoria

The Team:
Andrew Broekman *11089777*
Brenton Watt *14032644*
Fabio Loreggian *14040426*
Reinhardt Cromhout *14009936*

September 2016

Contents

1 Introduction

There are many cases in the industry, in research and in education where benchmarking of a software product is important. In industry benchmarks can be used to ensure that software performance is acceptable. In the competitive commercial world, benchmarks can serve to inform buyers of comparative performance of competing products. The benchmarking forms an integral part of research related to the development of new algorithms and techniques as well as the refinement and optimization of existing operations. Algorithm and data structure benchmarks can be applied as a very useful teaching tool for students to review the notions of space and time complexity.

2 Vision and Objectives

2.1 Vision

It is strange that although benchmarking seem to be a very common and useful application, very few benchmarking tools or services are available. Those that are available requires intricate configuration that may be beyond the reach of the developers, researchers, teachers and students who would like to use them. The development of a benchmarking service which can be used in a generic way would therefore be welcomed by a large potential user base.

2.2 Objectives

The main objectives of the benchmarking service is to provide an integrated platform:

- For benchmarking source code from various languages.
- For benchmarking various types of software code such as data structures, artificial intelligence algorithms, common application code and various other types of software code.
- Allowing users to compare the results of various benchmarks.
- To ease the management related to the generation, storing, cataloguing of test data, algorithm source code and measurements made by the system.

3 Overall Specifications

3.1 Functional Requirements

Most of the use cases follow all the same functional requirements specification. It can be assumed that all funcational requirements are mutatis mutandis the same unless otherwise stated by the respective use case in question.

3.1.1 Functional Requirements

The lower level services required by a use case service to either check the pre-conditions or address the post-conditions is shown in Figure 1

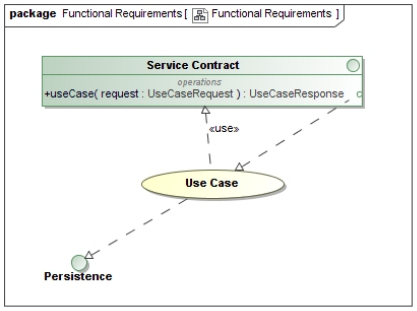

Figure 1: General Functional Requirements

4 Experiment Management

The *Experiment Management* module is responsible for the creation of an experiment with all it's associated jobs based on the specifications submitted by the user such as the algorithms, datasets, type of measurements, probe interval and number of iterations that should be performed.

This module further is responsible for communication with the message bus to place job requests on a queue, retrieve results from finished jobs and process the heartbeat messages of monitor nodes.

4.1 Scope

The scope for the Experiment Management Service module is shown in Figure 2.

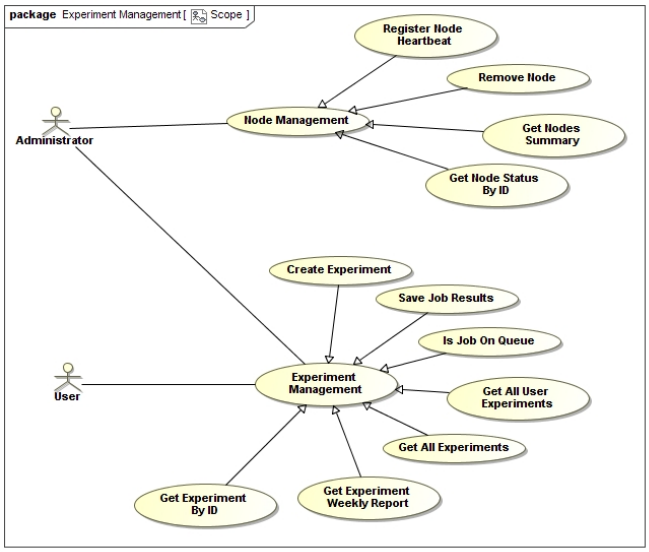

Figure 2: Experiment Management Scope

4.2 Domain Model

The domain model for the Experiment Management Service module is shown in Figure 3.

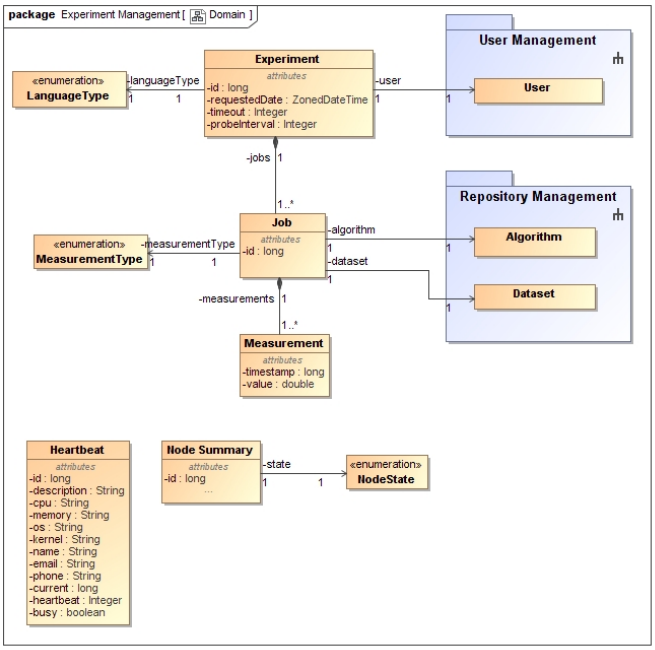

Figure 3: Experiment Management Domain Model

4.3 Create Experiment

The *Create Experiment* use case is concerned with creation of an experiment and its associated jobs specified by the supplied parameters. Upon creation the jobs will be placed at some pre-arranged location for monitor nodes to find.

4.3.1 Service Contract

The service contract for creating an experiment is shown in Figure 4.

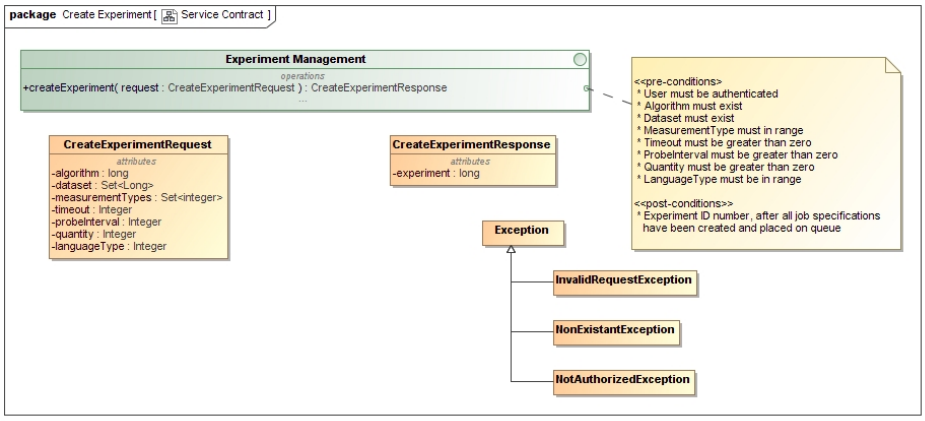

Figure 4: Create Experiment Service Contract

The use case creates a new experiment object as well as the associated jobs from the supplied parameters. The use case will match each algorithm and dataset to a measurement type. Such a matching constitutes one job for a set of parameters, after which a total of N jobs will be created for one such matching, where N is specified by the user quantity parameter.

4.3.2 Functional Requirements

The lower level services required by the create experiment service to either check the pre-conditions or address the post-conditions is shown in Figure 5.

Note that the create experiment service is responsible for randomizing the order of the jobs and placing them at the pre-arranged location for the monitor nodes to find. The other services

- retrieving the required objects and files to be sent along to the monitor nodes
- persist the experiment and jobs

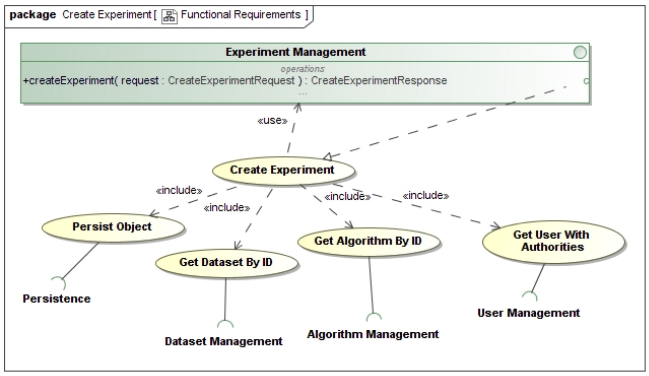

Figure 5: Create Experiment Functional Requirements

4.4 Get All Experiments

The *Get All Experiments* use case is concerned with retrieving a list of all experiments from the persistence provider.

4.4.1 Service Contract

The service contract for retrieving all experiments is shown in Figure 6.

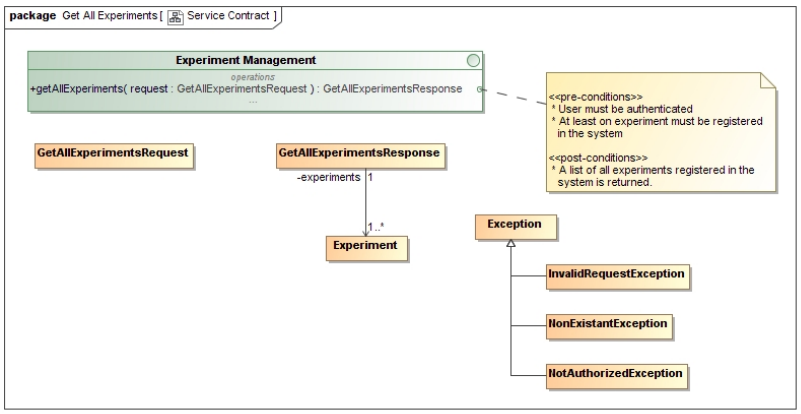

Figure 6: Get All Experiments Service Contract

4.5 Get All User Experiments

The *Get All User Experiments* use case is concerned with retrieving a list of all experiments related to the user in the current security context.

4.5.1 Service Contract

The service contract for retrieving all experiments for the currently logged-in user is shown in Figure 7.

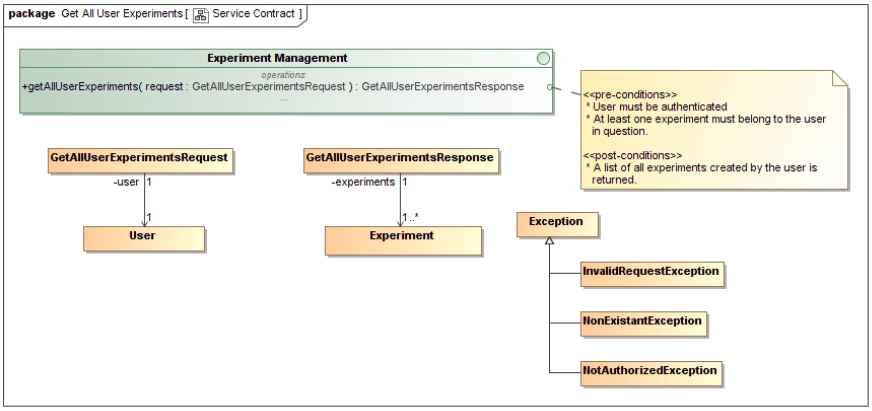

Figure 7: Get All User Experiments Service Contract

4.5.2 Functional Requirements

The lower level services required by the get all user experiments service to either check the pre-conditions or address the post-conditions is shown in Figure 8.

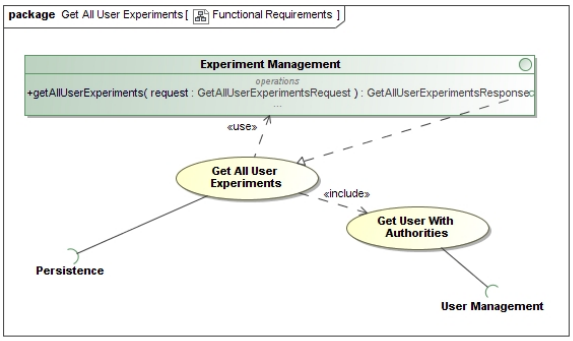

Figure 8: Get All User Experiments Functional Requirements

4.6 Get Experiment By ID

The *Get Experiment By ID* use case is concerned with retrieving an experiment based on its ID number.

4.6.1 Service Contract

The service contract for retrieving an experiment by its ID number is show in Figure 9.

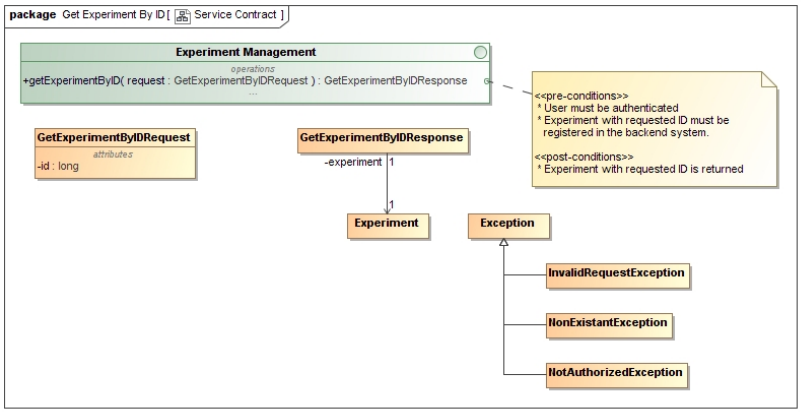

Figure 9: Get Experiment By ID Service Contract

4.7 Get Experiment Weekly Report

The *Get Experiment Weekly Report* use case is concerned with retrieving a range of metrics for all experiments done on the the benchmarking system over the last week (7 days).

4.7.1 Service Contract

The service contract for retrieving the weekly metrics of the management system is shown in Figure 10.

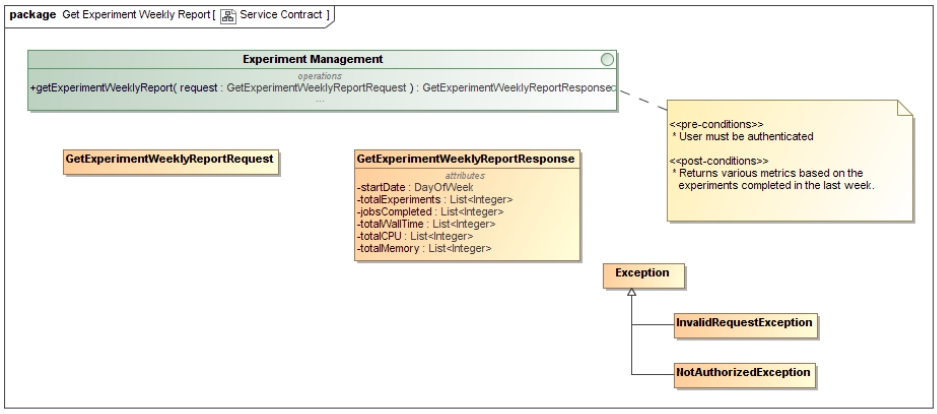

Figure 10: Get Experiment Weekly Report Service Contract

4.7.2 Functional Requirements

The lower level services required by the get experiment weekly report service to either check the pre-conditions or address the post-conditions is shown in Figure 11.

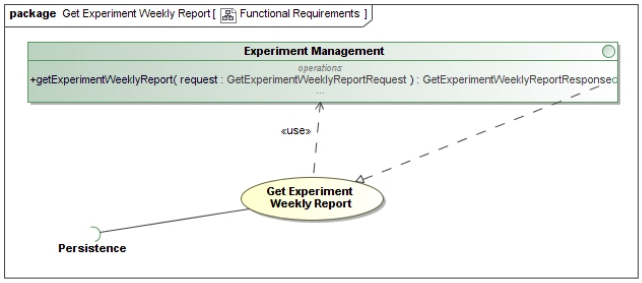

Figure 11: Get Experiment Weekly Report Functional Requirements

4.8 Get Node Status By ID

The *Get Node Status By ID* use case is concerned with retrieving the information, for a node identified by its ID number, currently stored in the monitoring system.

4.8.1 Service Contract

The service contract for retrieving the node information from the monitoring system is shown in Figure 12.

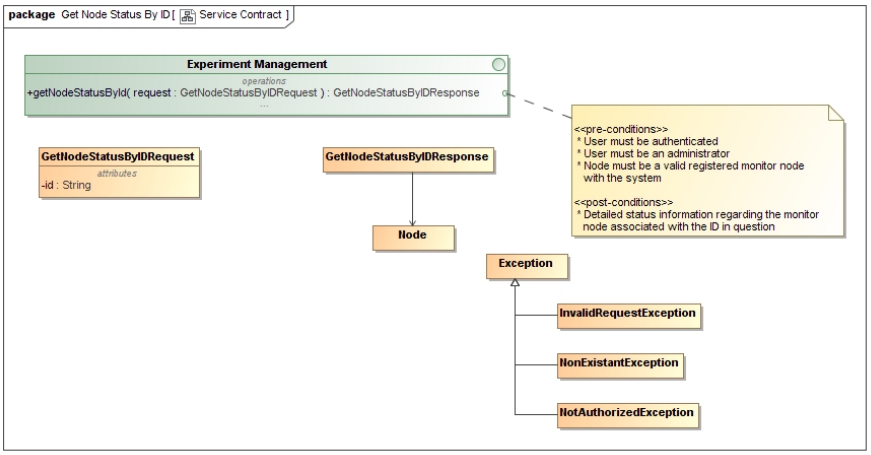

Figure 12: Get Node Status By ID Service Contract

4.8.2 Functional Requirements

The lower level services required by the get node status by ID service to either check the pre-conditions or address the post-conditions is shown in Figure 13.

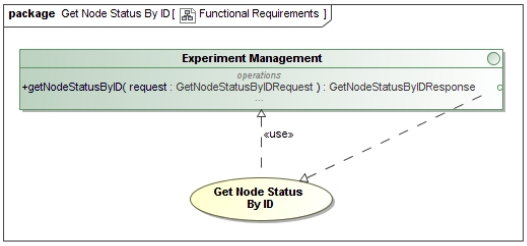

Figure 13: Get Node Status By ID Functional Requirements

4.9 Is Job On Queue

The *Is Job On Queue* use case is concerned with identifying if a job is currently still waiting on the queue to be processed, or if it is done being processed and hence have a list of measurements.

4.9.1 Service Contract

The service contract for checking if a specified job is still on the queue is shown in Figure 14.

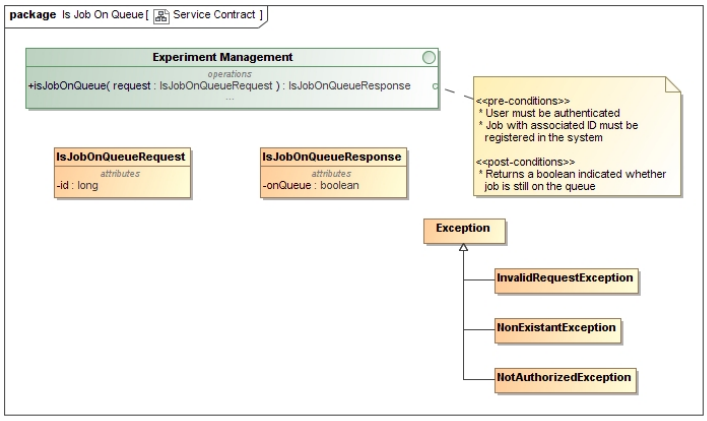

Figure 14: Is Job On Queue Service Contract

4.10 Register Node Heartbeat

The *Register Node Heartbeat* use case is concerned with registering the heartbeat of a monitor node with the monitoring system. This use case is used to both register new nodes and to refresh currently registered nodes.

4.10.1 Service Contract

The service contract for registering the heartbeat of a monitor node with the management system is shown in Figure 15.

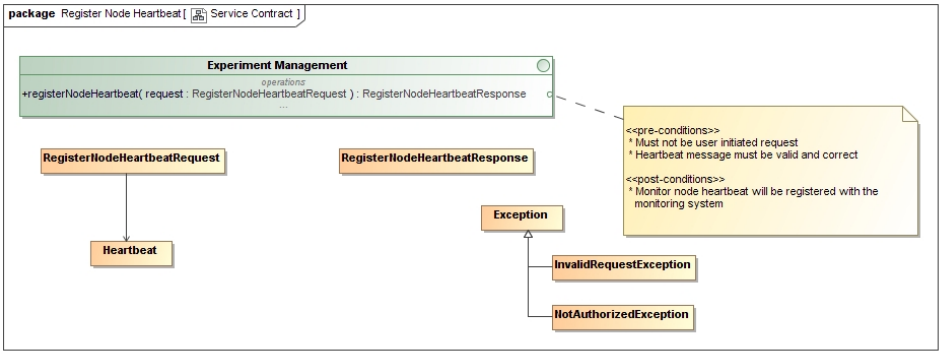

Figure 15: Register Node Heartbeat Service Contract

4.10.2 Functional Requirements

The lower level services required by the register node heartbeat service to either check the pre-conditions or address the post-conditions is shown in Figure 16.

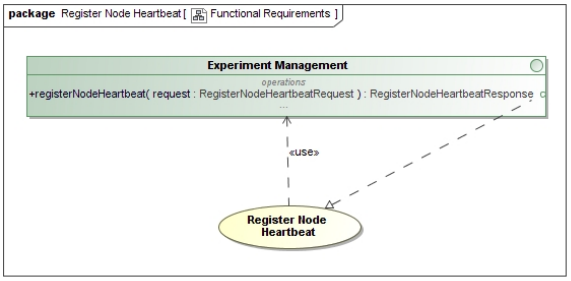

Figure 16: Register Node Heartbeat Functional Requirements

4.11 Remove Node

The *Remove Node* use case is concerned with removing a node from the monitoring system. Note that if the nodes continues to send heartbeat messages, the node will be re-registered in the system. This is to ensure a loose-coupling with the back end management system and the monitor nodes themselves that should remain totally autonomous and ephemeral. Further decoupling is accomplished by only allowing the nodes to communicate with AMQP queues.

4.11.1 Service Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 17.

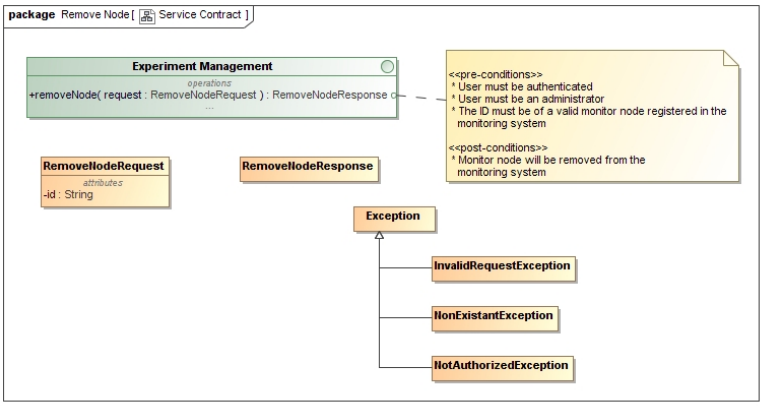

Figure 17: Remove Node Service Contract

4.11.2 Functional Requirements

The lower level services required by the remove node service to either check the pre-conditions or address the post-conditions is shown in Figure 18.

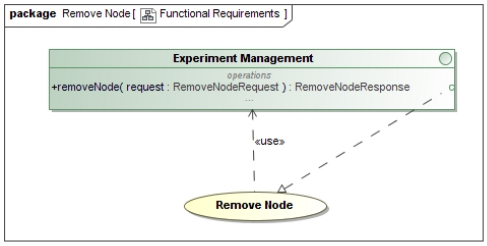

Figure 18: Remove Node Functional Requirements

4.12 Save Job Results

The *Save Job Results* use case is concerned with removing results from a pre-arranged central location with the monitor nodes and associating the results to the experiment in the back end.

4.12.1 Service Contract

The service contract for retrieving the saving the results for a job is shown Figure 19.

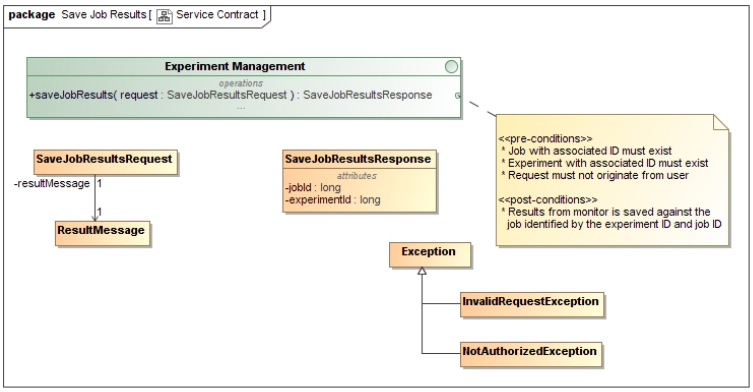

Figure 19: Save Job Results Service Contract

5 Notification

The *Notification* module is currently used for sending notifications to the user upon account creation or when the user tries to recover a lost account. The module can however be expanded in future to allow other notification services and hence the decision to include this as a major module in the system and not as a submodule of the User Management module.

5.1 Scope

The scope for the Notification modules is shown in Figure 20.

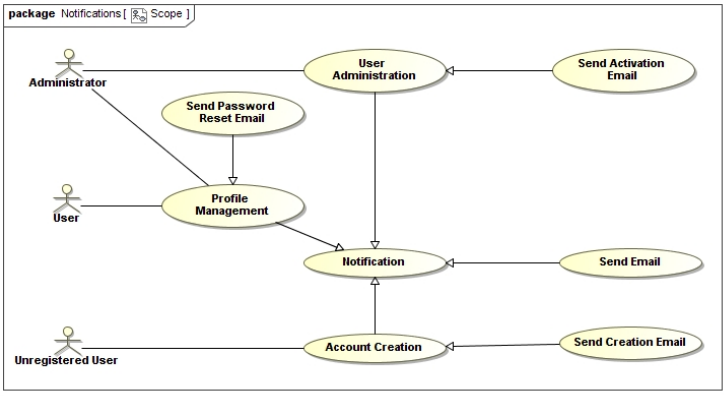

Figure 20: Notification Scope

These use cases will be used by the User Management Module to assist with the user

authentication and recovering of lost accounts. The module however provides general functionality to send emails which can be used by any other module that may require this in future.

5.2 Send Activation Email

The *Send Activation Email* use case is concerned with with sending a new created managed user an email on how to setup there password. Important to note is that the user's account has been activated as it was created by an administrator, however the administrator should not have control off selecting a password for the user as this will violate general security principles.

5.2.1 Service Contract

The service contract for sending an activation email upon account creation is shown in Figure 21

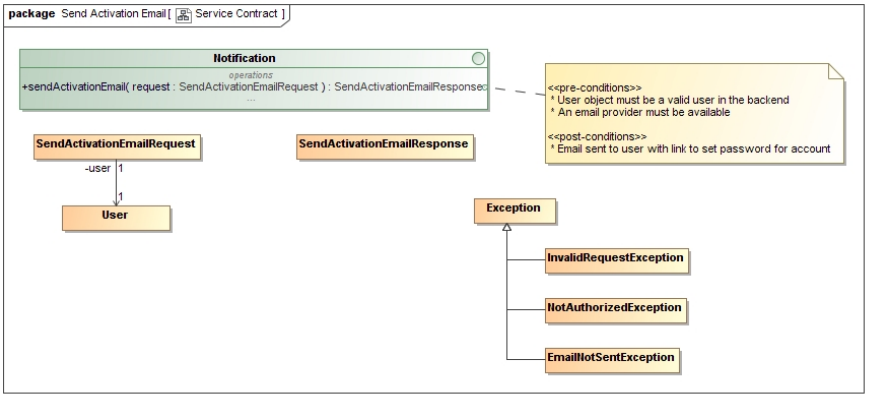

Figure 21: Send Activation Email Service Contract

5.2.2 Functional Requirements

The lower level services required by the send activation email service to either check the pre-conditions or address the post-conditions is shown Figure 22.

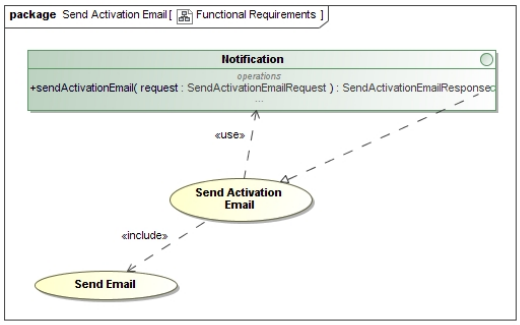

Figure 22: Send Activation Email Functional requirements

5.3 Send Creation Email

The *Send Creation Email* use case is concerned with with sending a new unmanaged user an email on how to activate their account.

5.3.1 Service Contract

The service contract for send a creation email is shown in Figure 15.

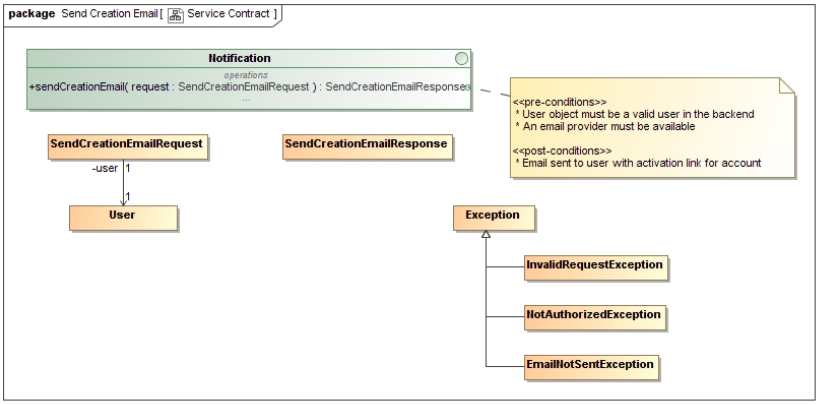

Figure 23: Send Creation Email Service Contract

5.3.2 Functional Requirements

The lower level services required by the send creation email service to either check the pre-conditions or address the post-conditions is shown Figure 24.

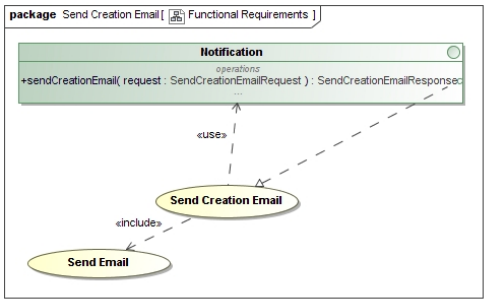

Figure 24: Send Creation Email Functional Requirements

5.4 Send Email

The *Send Email* use case is concerned with with sending a body of content to a specified receipt. The use case can be used as is, but it is advised to rather make use of one of the wrapper use cases.

5.4.1 Service Contract

The service contract for sending an email is shown in Figure 15.

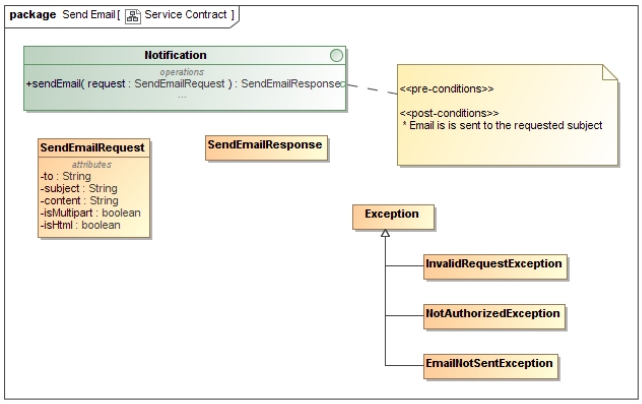

Figure 25: Send Email Service Contract

5.4.2 Functional Requirements

The lower level services required by the send email service to either check the pre-conditions or address the post-conditions is shown Figure 26.

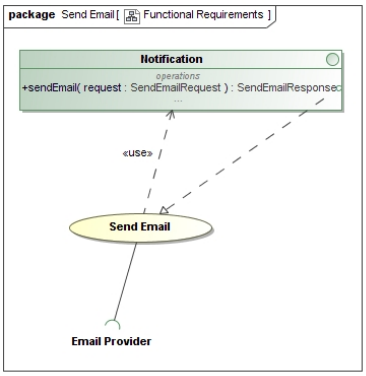

Figure 26: Send Email Functional Requirements

5.5 Send Password Reset Email

The *Send Password Reset Email* use case is concerned with with sending a user instructions on how to go about to reset there lost account password.

5.5.1 Service Contract

The service contract for sending a password reset email is shown in Figure 15.

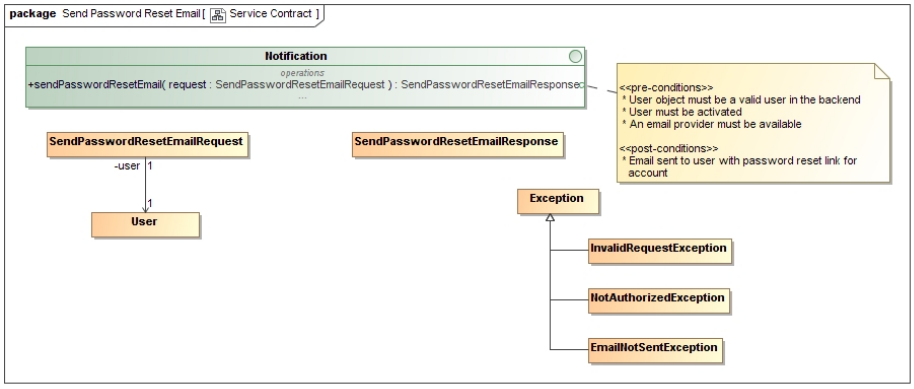

Figure 27: Send Password Reset Email Service Contract

5.5.2 Functional Requirements

The lower level services required by the send password reset email service to either check the pre-conditions or address the post-conditions is shown Figure 28.

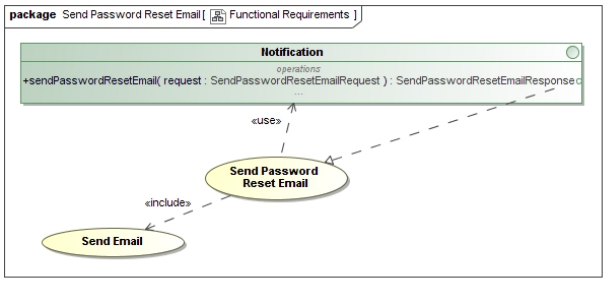

Figure 28: Send Password Reset Email Functional Requirements

6 Reporting

The Reporting module represents the core of the value that the users will get out of the system. Reports are the means with which the users will be able to see the results of the benchmarking of their programs.

6.1 Scope

The scope for the reporting module is shown in Figure 29

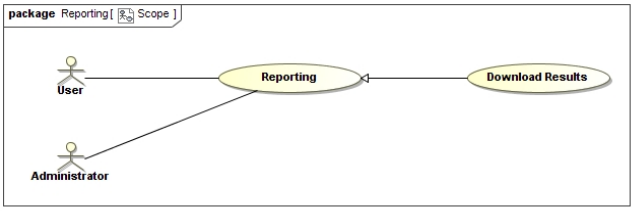

Figure 29: Reporting Scope

6.2 Download Results

The *Download Results* use case is concerned with presenting the results of a job to the user in a global interchange format, which was chosen as CSV (Comma Seperated Value) representation.

6.2.1 Service Contract

The service contract for downloading results for a specified job is shown in Figure 30

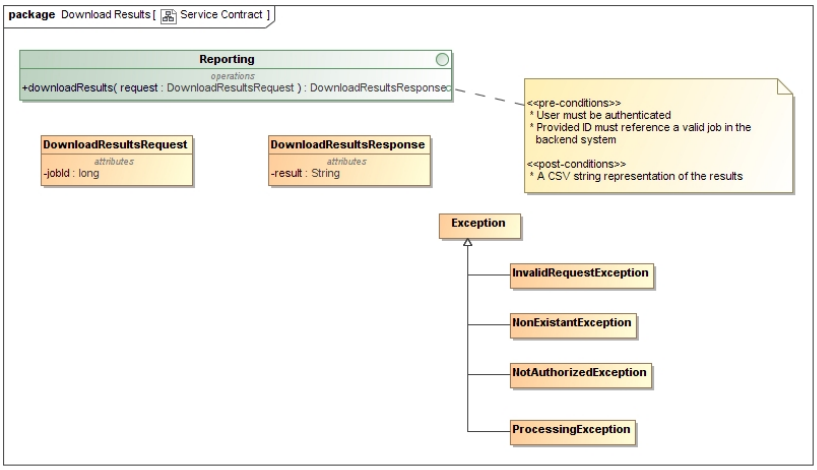

Figure 30: Download Results Service Contract

7 Repository Management

The *Repository Management* module is responsible for management and cataloging of algorithms and test data to be used in the benchmarking tests.

7.1 Domain Model

The domain model for the Repository Management module is shown in Figure 31

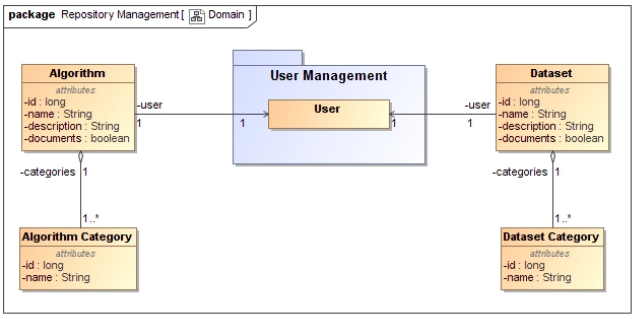

Figure 31: Repository Management Domain Model

7.2 Scope

The scope for the Repository Management module is complex because of the constraints placed upon the use cases by the business requirements. Therefore the module scope has

been broken down into different levels of granularity to clarify the business requirements. A high level overview of the scope is given in Figure 32

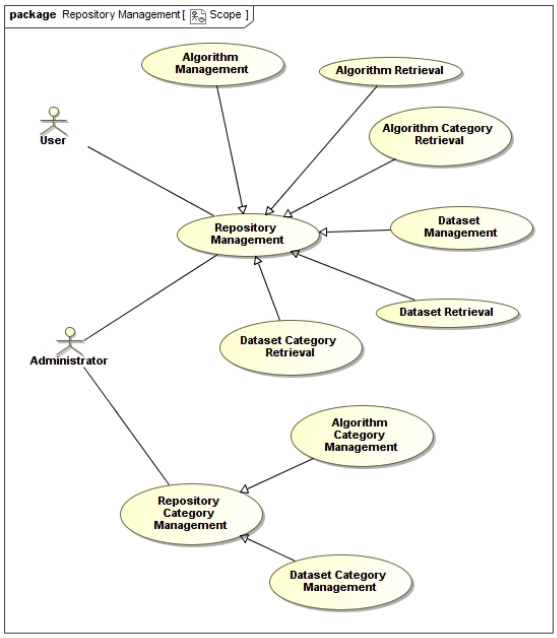

Figure 32: High Level Scope overview of Repository Management

For the dataset use cases four major scopes have been identified:

- Dataset Management
- Dataset Retrieval
- Dataset Category Management
- Dataset Category Retrieval

7.2.1 Dataset Management

The dataset management scope is concerned with all responsibilities related to the CRUD management of datasets in the repository. Important to note, that the dataset it self can't be updated, but only the metadata associated with a dataset can be updated. This is done to ensure consistency in the system, as it was stated by a client, that any modification to a dataset constitutes a new entity.

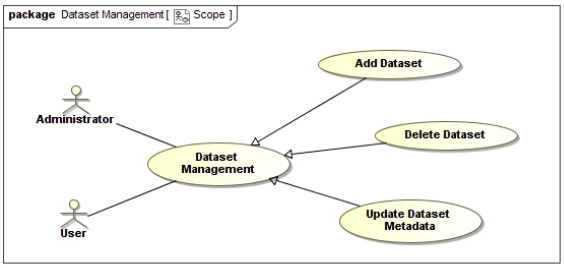

Figure 33: Scope overview of Dataset Management

7.2.2 Dataset Retrieval

This scope details the various retrieval mechanism by which datasets can be retrieved.

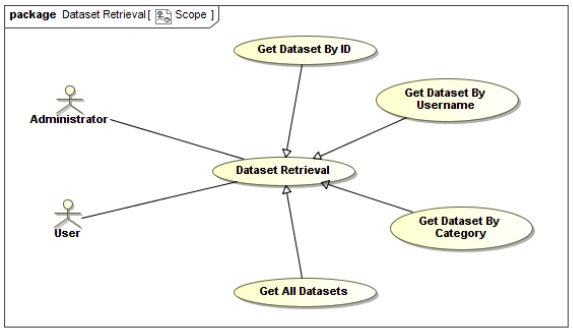

Figure 34: Scope overview of Dataset Retrieval

7.2.3 Dataset Category Management

The CRUD management of dataset classifiers is limited only to administrator users as the client want to prevent the pollution of the system with user created categories.

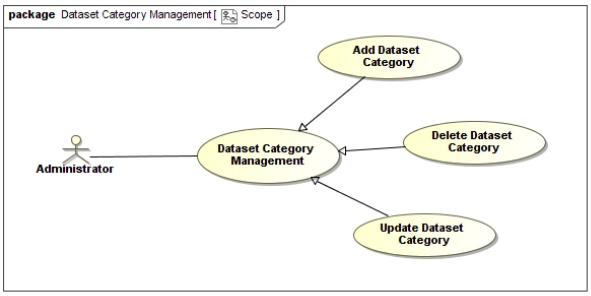

Figure 35: Scope overview of Dataset Category Management

7.2.4 Dataset Category Retrieval

This scope details the various retrieval mechanism by which dataset classifiers can be retrieved.

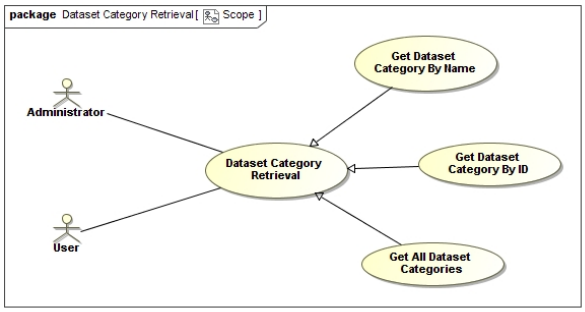

Figure 36: Scope overview of Dataset Category Retrieval

7.2.5 Algorithm and Algorithm Category Scope

The algorithm and algorithm category scopes is mutatis mutandis the same as for the dataset and dataset categories.

7.3 Add Dataset

The *Add Dataset* use case is concerned with adding a user uploaded dataset to the repository management system. This will allow the dataset to be used in the creation of experiments.

7.3.1 Service Contract

The service contract for adding a dataset is shown in Figure 37

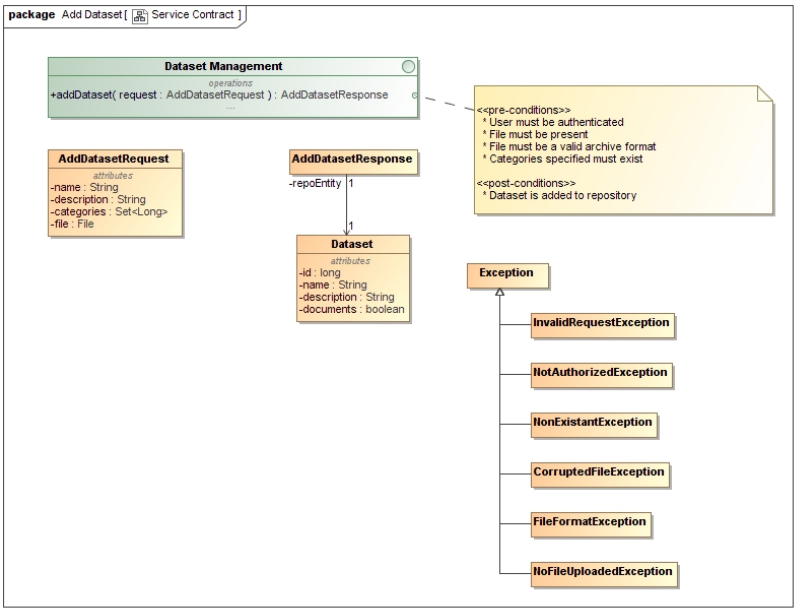

Figure 37: Add Dataset Service Contract

The use case adds a user uploaded dataset to the management system. The service is responsible for adding the metadata object to the relational persistence provider and adding the uploaded archive to the document-based persistence provider. Furthermore this use case is also responsible for extracting the archive and storing each document in the document-based persistence provider with a unique identifier, as to allow the user to request individual documents from an uploaded archive.

7.3.2 Functional Requirements

The lower level services required by the add dataset service to either check the pre-conditions or address the post-conditions are shown in Figure 38

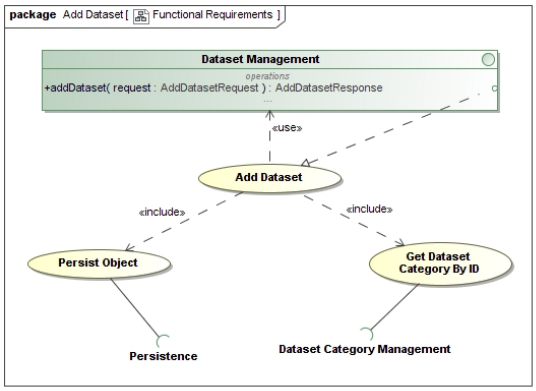

Figure 38: Add Dataset Functional Requirements

7.3.3 Process Design

Refer to figure 39 on the process flow to add a new user uploaded dataset to the benchmarking system. Note that the Get user With Authorities and Get Dataset Category By ID are all responsible for checking whether the obejct exists.

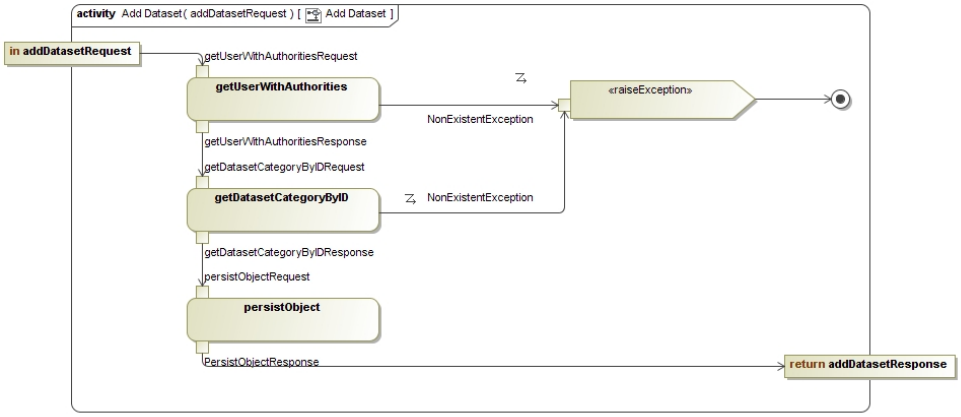

Figure 39: Add Dataset Process Design

7.4 Delete Dataset

The *Add Dataset* use case is concerned with removing a user uploaded dataset from the repository management system, however a dataset can only be removed if it has not yet been used in an experiment definition.

7.4.1 Service Contract

The service contract for deleting a dataset is shown in Figure 40

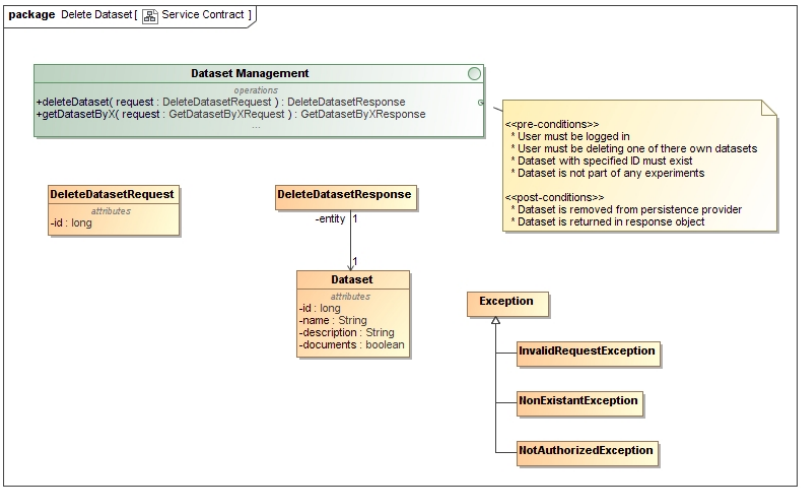

Figure 40: Delete Dataset Service Contract

7.4.2 Functional Requirements

The lower level services required by the delete dataset service to either check the pre-conditions or address the post-conditions are shown in Figure 41

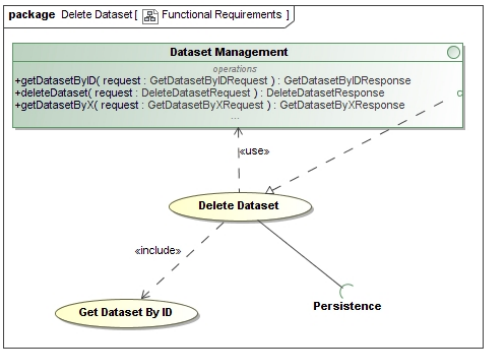

Figure 41: Delete Dataset Functional Requirements

7.4.3 Process Design

Refer to figure 42 on the process flow to delete an uploaded dataset from the benchmarking system. Note that the Get user With Authorities and Get Dataset Category By ID

are all responsible for checking whether the obejct exists. The Delete Dataset service is responsible for ensuring that the user requesting the delete can only delete their own uploaded datasets.

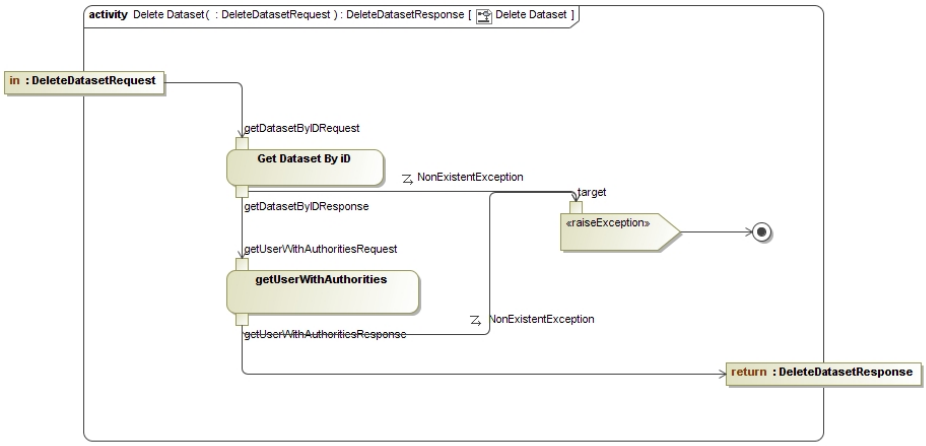

Figure 42: Delete Dataset Process Design

7.5 Update Dataset Metadata

The *Update Dataset Metadata* use case is concerned with updating the metadata around a user uploaded dataset. Important to note that the actual dataset may not and can not be updated, as this will violate the integrity of results.

7.5.1 Service Contract

The service contract for updating a datasets metadata is shown in Figure 43

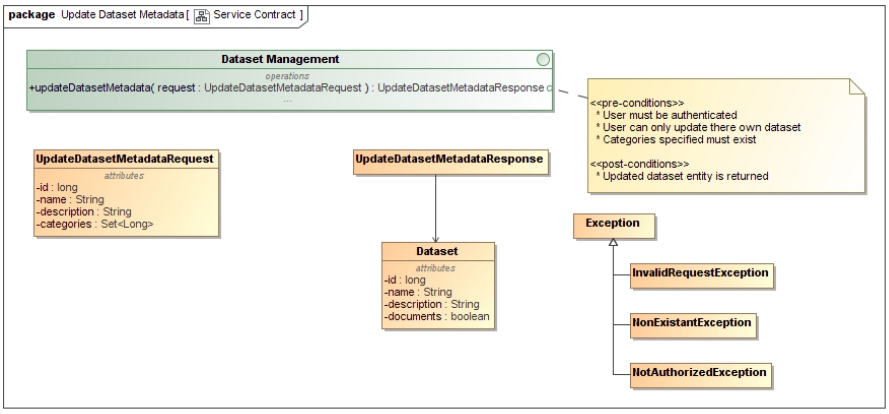

Figure 43: Update Dataset metadata Service Contract

7.5.2 Functional Requirements

The lower level services required by the update dataset metadata service to either check the pre-conditions or address the post-conditions are shown in Figure 44

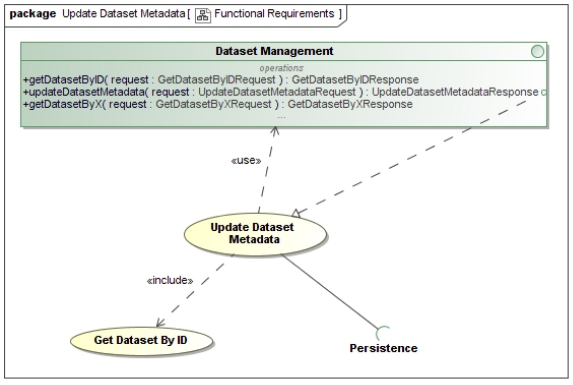

Figure 44: Update Dataset Metadata Functional Requirements

7.6 Get Dataset By X

3 similar use case have been defined to retrieve the dataset upon namely

- Get Dataset By Category
- Get Dataset By ID
- Get Dataset By Username

All service contracts, functional requirements are implementations are mutatis mutandis the same, except the parameter which is filtered upon. The respective request and response objects can be seen in figure 45.

7.6.1 Service Contract

The generic service contract for retrieving a dataset upon a parameter is show in Figure 45

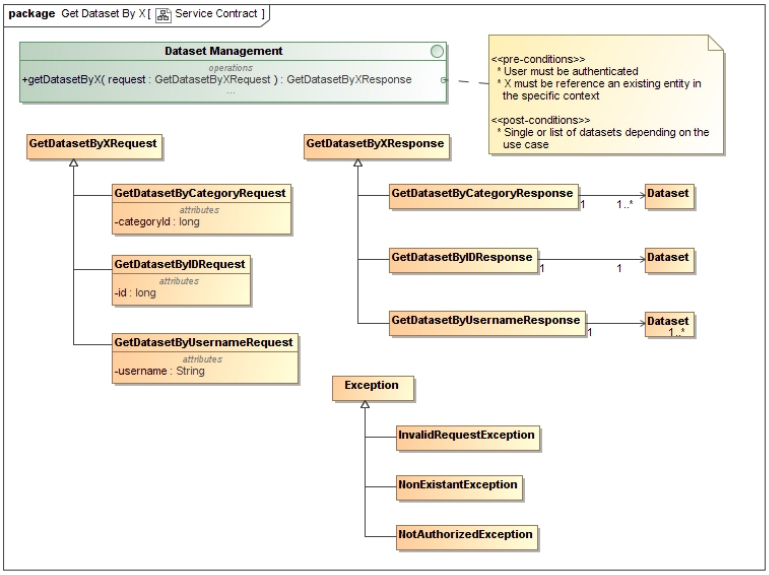

Figure 45: Get Dataset by X Service Contract

7.6.2 Functional Requirements

The lower level services required by the generic get dataset by X service to either check the pre-conditions or address the post-conditions are shown in Figure 46

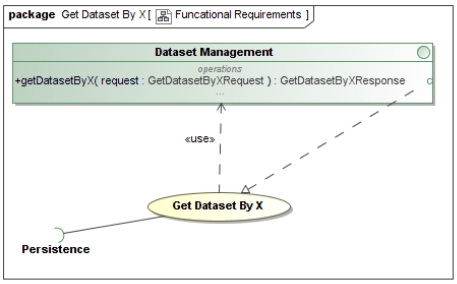

Figure 46: Get Dataset by X Functional Requirements

7.7 Get All Datasets

The *Get All Datasets* use case is concerned with retrieving all datasets from the back end system.

7.7.1 Service Contract

The service contract for retrieving all datasets is shown in Figure 47.

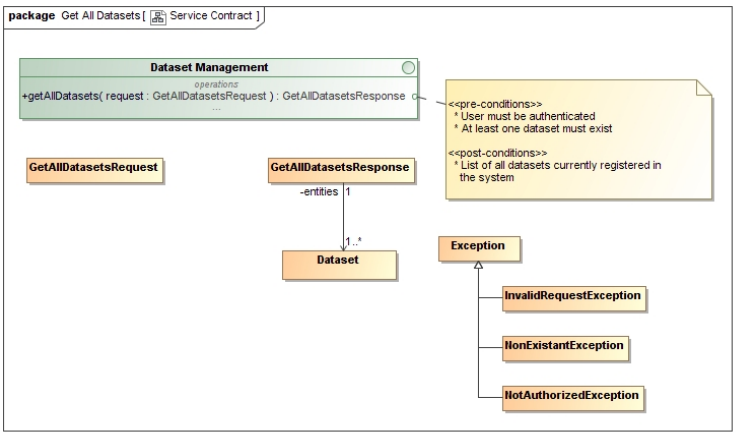

Figure 47: Get All Datasets Service Contract

7.8 Algorithm Management

The algorithm management service contract and implementation is mutatis mutandis the same as that of the dataset management.

7.9 Add Dataset Category

The *Add Dataset Category* use case is concerned with adding a new dataset category that can be used to classify datasets.

7.9.1 Service Contract

The service contract for adding a dataset category is shown in Figure 48.

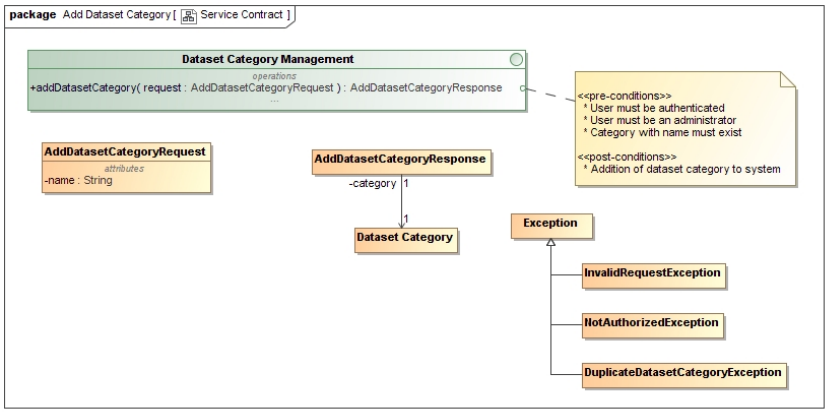

Figure 48: Add Dataset Category Service Contract

7.9.2 Functional Requirements

The lower level services required by the update dataset metadata service to either check the pre-conditions or address the post-conditions are shown in Figure 49.

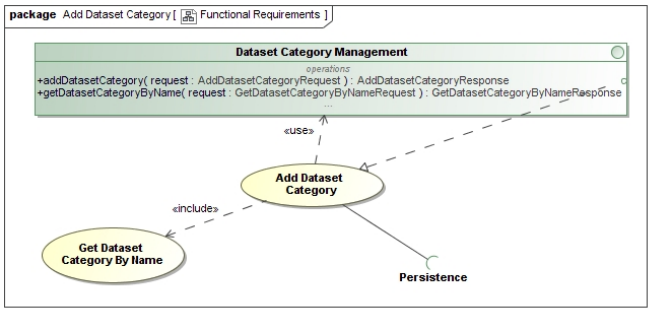

Figure 49: Add Dataset Category Service Contract

7.10 Delete Dataset Category

The *Delete Dataset Category* use case is concerned with removing a dataset category from the back end.

7.10.1 Service Contract

The service contract for deleting a dataset category is shown in Figure 50.

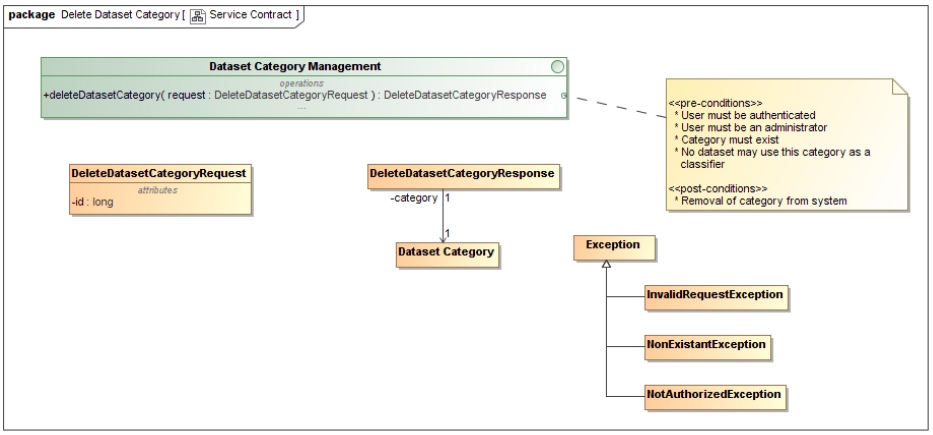

Figure 50: Delete Dataset Category Service Contract

7.11 Update Dataset Category

The *Update Dataset Category* use case is concerned with updating a dataset category.

7.11.1 Service Contract

The service contract for updating a dataset category is shown in Figure 51.

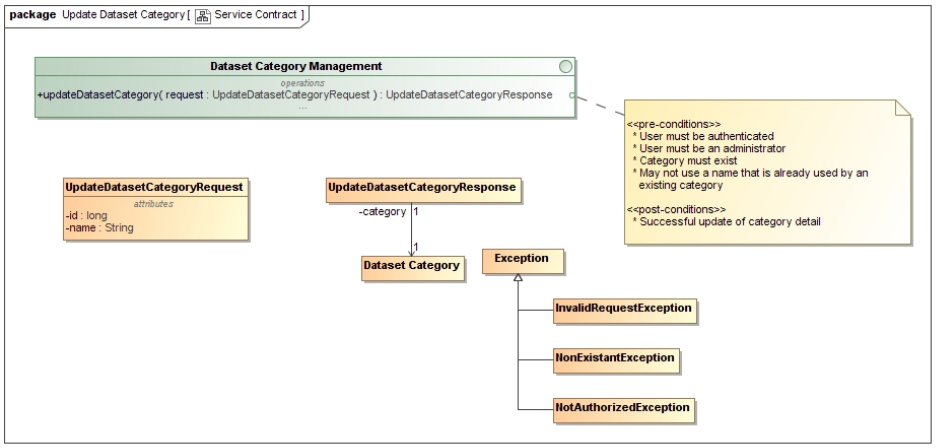

Figure 51: Update Dataset Category Service Contract

7.11.2 Functional Requirements

The lower level services required by the update dataset metadata service to either check the pre-conditions or address the post-conditions are shown in Figure 52.

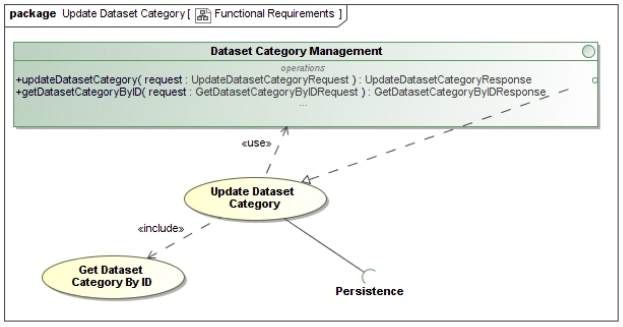

Figure 52: Update Dataset Category Service Contract

7.12 Get Dataset Category By X

2 similar use case have been defined to retrieve the dataset category upon namely

- Get Dataset Category By ID
- Get Dataset Category By Name

All service contracts, functional requirements are implementations are mutatis mutandis the same, except the parameter which is filtered upon. The respective request and response objects can be seen in Figure 53.

7.12.1 Service Contract

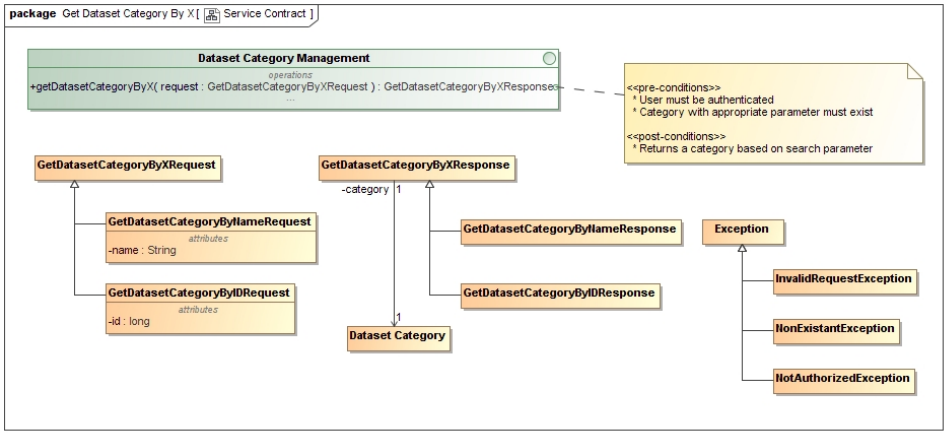

Figure 53: Get Dataset by X Service Contract

7.12.2 Functional Requirements

The lower level services required by the update dataset metadata service to either check the pre-conditions or address the post-conditions are shown in Figure 54.

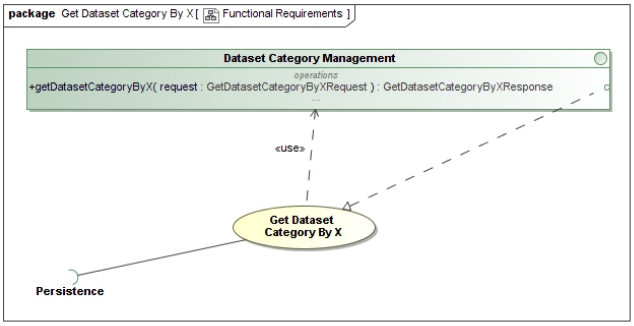

Figure 54: Get Dataset by X Functional Requirements

7.13 Get All Dataset Categories

The *Get All Dataset Categories* use case is concerned with retrieving a list of all dataset categories in the back end.

7.13.1 Service Contract

The service contract for getting all dataset categories is shown in Figure 55.

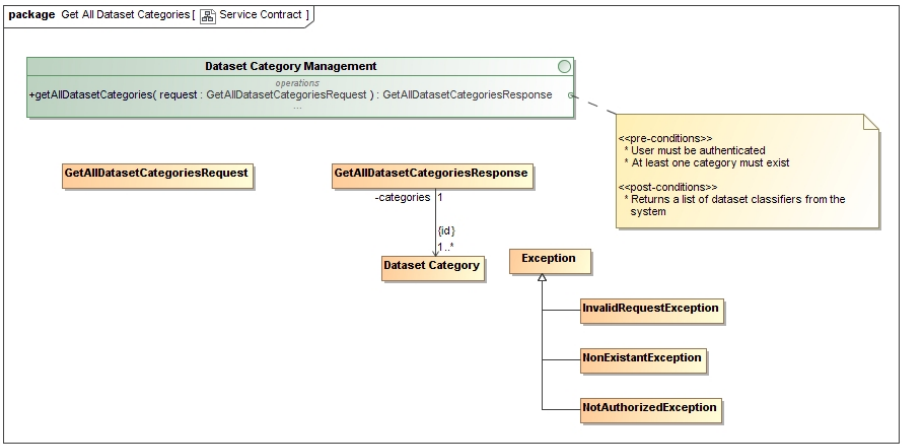

Figure 55: Get All Dataset Categories Service Contract

7.14 Algorithm Category Management

The algorithm category management service contract and implementation is mutatis mutandis the same as that of the dataset category management.

8 User Management

The *User Management* module is responsible for maintaining demographic information about the registered users of the system, including the Authority levels of each user.

8.1 Scope

The scope for the users module is shown in Figure 8.1

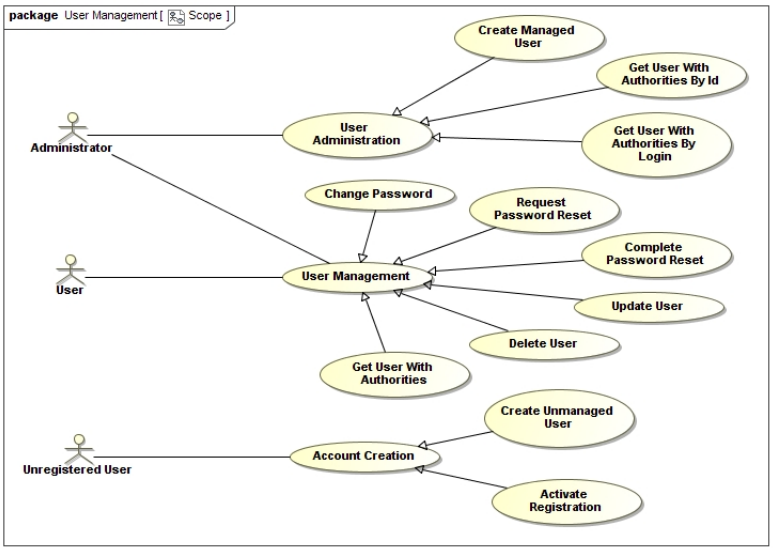

Figure 56: Users Scope

The scope of the users module include:

- Administrator users can add and remove Regular Users
- Any person who wants use the benchmarking service can register to the system and become a user. They can then also edit their profile.

8.2 Domain Model

The domain model for the users module is shown in Figure 8.2

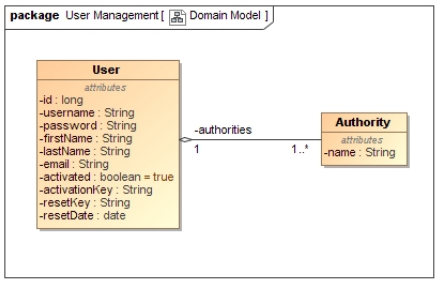

Figure 57: Users Domain Model

There are 2 main types of users who are at different authority levels. *Administrator* users will have the permissions needed to manage the website, manage regular users and

do certain administrative duties on the system.

Users can be registered by anyone who wants to use the benchmarking service. These users will be able to conduct benchmarks but will not be able to preform any administrative duties on the system.

8.3 Activate Registration

The *Activate Registration* use case is concerned with activating the account of a user created by the *Create Unmanaged User* service.

8.3.1 Services Contract

The service contract for activating a user account is shown in Figure 58.

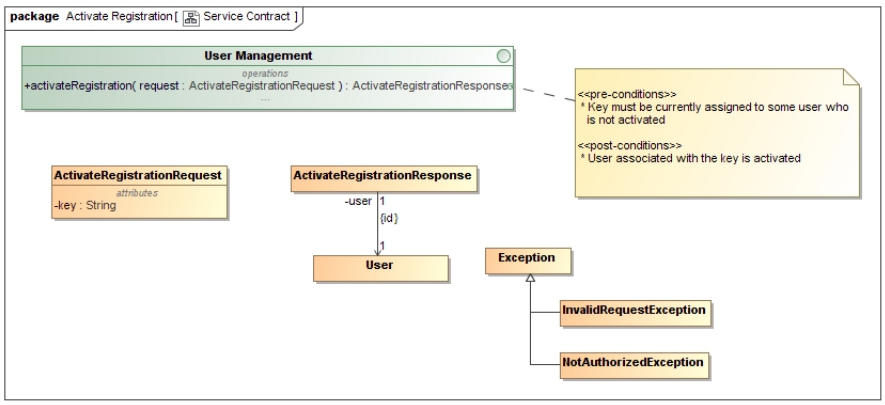

Figure 58: Activate Registration Service Contract

8.4 Change Password

The *Change Password* use case is concerned with allowing the currently logged-in user to change their password.

8.4.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 17.

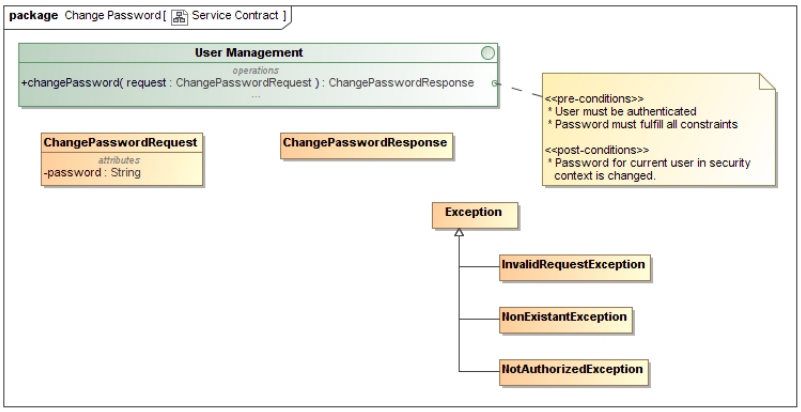

Figure 59: Change Password Service Contract

8.5 Complete Password Reset

The *Complete Password Reset* use case is concerned with setting the new password of a lost account. This is the final step to complete on an account after the *Request Password Reset* service has been called first on the lost account.

8.5.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 17.

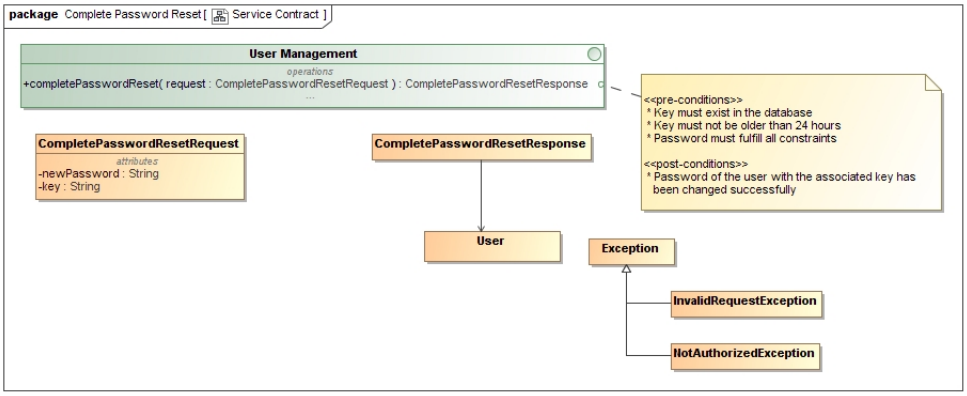

Figure 60: Complete Password Reset Service Contract

8.6 Create Managed User

The *Create Managed User* use case is concerned with allowing an administrator to create a user account deferring the selection of password to the final user.

8.6.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 61.

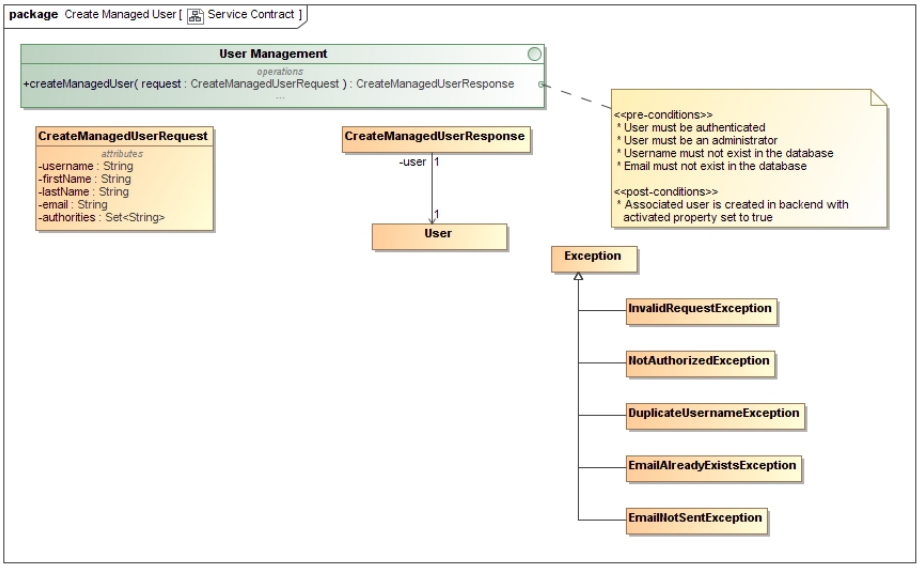

Figure 61: Create Managed User Service Contract

8.7 Create Unmanaged User

The *Create Unmanaged User* use case is concerned with allowing a user to create for themselves an account on the benchmark system, with a default role of *User*.

8.7.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 62.

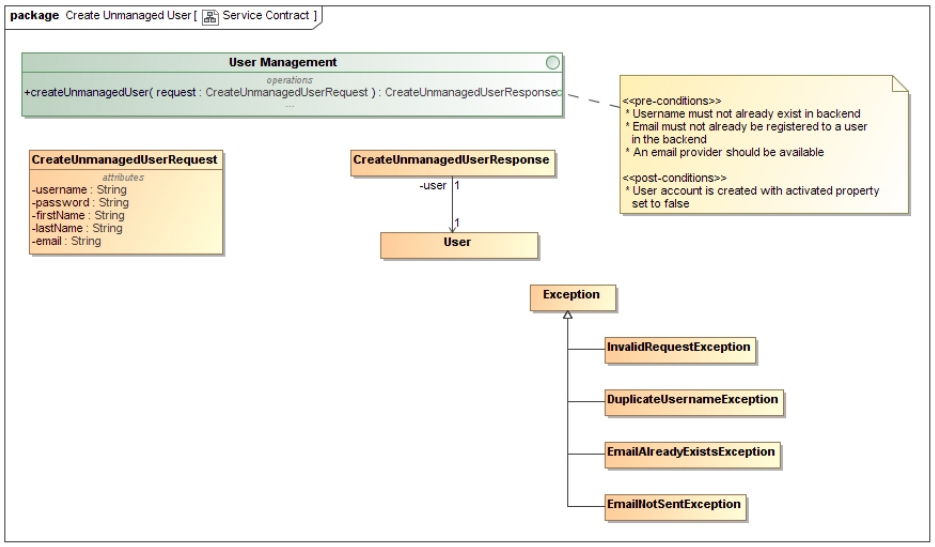

Figure 62: Create Unmanaged User Service Contract

8.7.2 Process Design

Refer to figure 63 on the process flow to create a new unmanaged user account, i.e. the end user creates there own account. Note that the Get user With Authorities service is used to confirm if another user with that same username already exists in the system, if so an DuplicateUsernameException is raised. Next the Create Unmanaged User user is responsible for confirming that no account is already associated with the email address passed in with the request object. If email address is already associated with an account a EmailAlreadyExistsException is raised. After the account has been created a notification will be sent to the user. If email could not be sent to the user, a EmailNotSentException is raised.

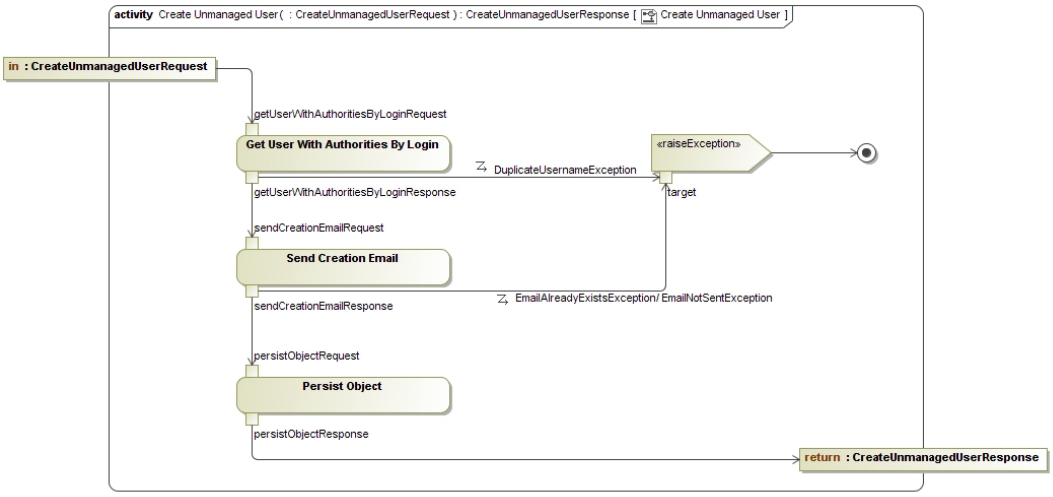

Figure 63: Create Unmanaged User Process Design

8.8 Delete User

The *Delete User* use case is concerned removing a user from the back end system.

8.8.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 64.

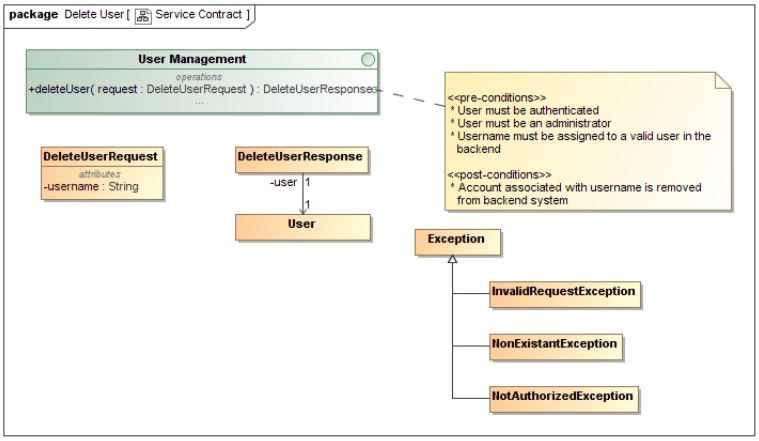

Figure 64: Delete User Service Contract

8.9 Get User With Authorities

The *Get User With Authorities* use case is concerned with retrieving the current user with associated roles from the current security context.

8.9.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 65.

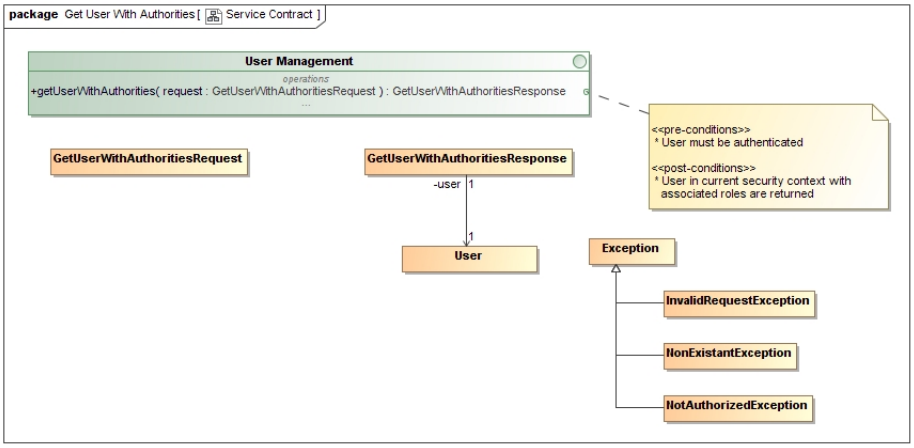

Figure 65: Get User With Authorities Service Contract

8.10 Get User With Authorities By Login

The *Get User With Authorities By Login* use case is concerned with retrieving a specified user with there associated roles from the back end.

8.10.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 66.

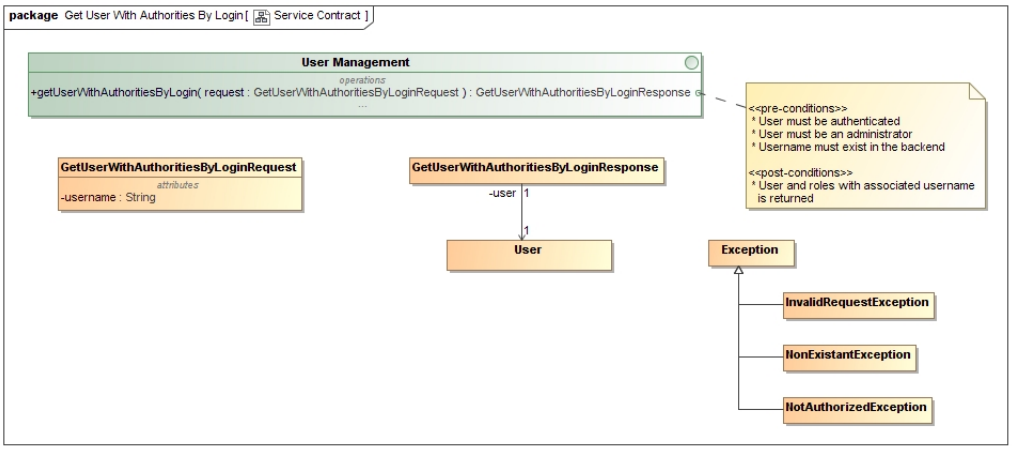

Figure 66: Get User With Authorities By Login Service Contract

8.11 Request Password Reset

The *Request Password Reset* use case is concerned with allowing a user to retrieve a lost account. This is the first step in account retrieval, with the second and final step being the *Complete Password Reset* service.

8.11.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 67.

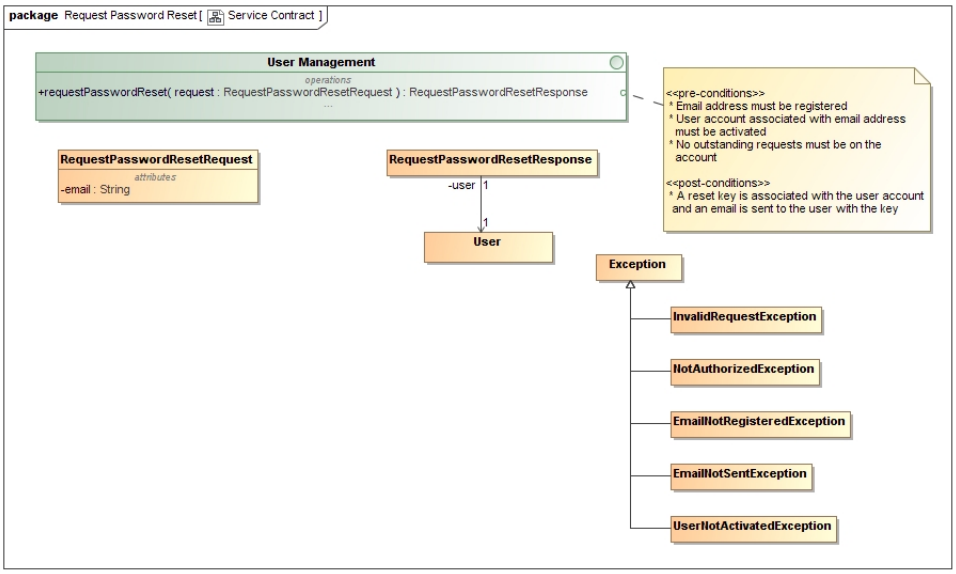

Figure 67: Request Password Reset Service Contract

8.12 Update User

The *Update User* use case is concerned with allowing the currently logged-in user to update their associated profile data. Important to note is that the user is not able to update their username.

8.12.1 Services Contract

The service contract for removing a registered monitor node from the back end system is shown in Figure 68.

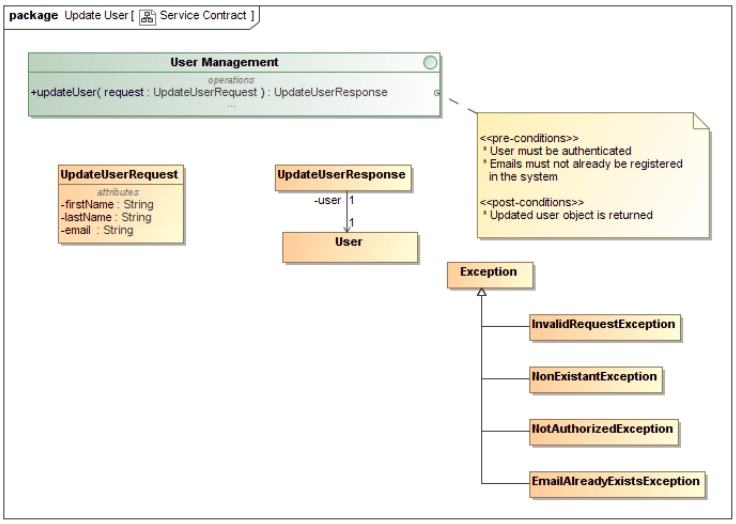

Figure 68: Update User Service Contract

8.12.2 Functional Requirements

The lower level services required by the create experiment service to either check the pre-conditions or address the post-conditions is shown in Figure 69.

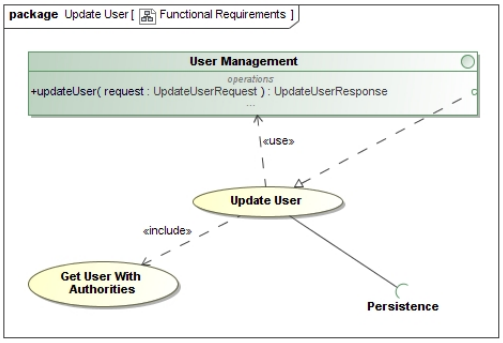

Figure 69: Update User Functional Requirements