University of Pretoria

COS 301 - Software Engineering

The Savage Ru's

Software Requirements Specification and Technology Neutral Process Design

Author(s):	*Student number(s):*
Jodan Alberts	14395283
Mark Klingenberg	14020272
Una Rambani	14004489
Ruan Klinkert	14022282

May 27, 2016

Contents

1 Introduction

This is the software requirements specification for the vizARD Augmented Reality application being developed for EPI-USE Labs by The Savage Ru's.

VizARD is a mobile application which will allow a user to take a picture of tabulated data and then view, automatically generated, 3D graphs of the data projected onto the document of which the image was taken.

The document you are reading is structured as follows:

- **Vision** - this includes information on the purpose of the system, from the client's perspective. Possible users, uses and results that they intend to get from VizARD once it launches. As well as problems they intend for the app to solve.
- **Background** - this contains information on the inspiration for the application. Which gap in the market the client intends to fill and what inspired the development of VizARD.
- **Software Architecture** - here we discuss the architecture of the system. This includes requirements and constraints as well as technological decisions and other non-functional aspects of VizARD.
- **Functional requirements and application design** - this section handles the use-cases and overall functionality of the system.
- **Open Issues** - there are inevitably questions that remain to be answered regarding the VizARD system and they are listed in this section. Problems here will be handled as they arise further in the development cycle.

2 Vision

EPI-USE Labs (henceforth referred to as "the client") intends for the VizARD application to be used by a large variety of mobile device users across both Android and iOS platforms. VizARD helps to simplify the analysis of numerical data through visualization, in the form of automatically generated 3D graphs.

Fundamentally, the system will allow a user to take a picture of a table of numerical data which he/she may need to interpret. The application will then use OCR (Optical Character Recognition) to read the data from the picture. It will then decide on an appropriate graph for the type of data and generate a graph for the data. After the graph is generated, it will project a 3D model of the graph onto the image (or, ideally, onto a live stream of the paper) for the user to view.

Additionally, the system will allow users to send images (or screen captures) of generated graphs to other devices via popular social media channels.

Typically usage will be as follows:

- The user (possibly a businessman) finds tabular data he/she would like to analyse more easily.
- The user opens the app.
- Once the app is open and loaded, the user takes a picture of the table he/she would like to analyse.
- The user receives a notification that the graph has been generated and the generated graph is displayed on the screen (mapped onto the paper).
- The user taps on the "Share" button and is presented with several options through which he/she can share the graph.
- An option is selected and an image of the graph is sent to the other user.

3 Background

It is much simpler for us to recognize patterns and make quick analysis of data if it is presented to us in visual form. A simple example for the use of such an application would be a principal at a school who is presented with the Mathematics results of a particular grade for several quarters, such an application would make it very simple for him to quickly visualize the numeric data and see the trend.

The problem at hand is that there is a lot of information to go around and so little time to process. In a society that demands us to make decisions quickly, it would be wise to have a tool that aids the decision making process by making the information easier to digest and that is what vizARD intends to do.

Potential users could range from students, researchers, people in business, managers at stores and anyone else who would like to visualize data on the go.

4 Software Architecture

In this section we discuss the software architecture, including architecture scope, requirements, access and integration requirements, quality requirements and architectural constraints.

4.1 Architectural Scope

The ViZARD app will be an offline application (not connected directly to any network/server) which will allow a user to take a picture - using a cellphone's built-in camera - and generate a 3D graph from information in the image. Tesseract - an open source OCR (Optical Character Recognition)- is used to evaluate the image and find relevant information for generating the graph. Unity 3D and OpenGL will be used to generate a graph of the information and finally Vuforia AR (Augmented Reality) SDK is used to project the graph onto a image marker for viewing. Furthermore users will have the ability to share graphs via several social applications on their cellphone/tablet. And finally all these systems will be running on Android OS and iOS based devices.

4.2 Integration Requirements

The VizARD app will integrate with Android OS, and iOS, and use the suite of Android APIs which accompany the OSes. Specifically, APIs will be used to integrate with the sharing functions in order to share to different social media platforms and messaging apps. Additionally the app will gain access to the file system and camera via the built-in APIs. Furthermore, Java APIs will be used to integrate between the various systems which make up VizARD's functionality. For instance between OpenCV and Tesseract for the OCR.

4.3 Access Channel Requirements

The system will function as an offline application. As such there will not be any outside system with direct access to the application. There are, however, access channels for users. Specifically users will gain access to the system via two mobile device operating system:

- Android OS
- iOS

Furthermore any data that must be sent between the Operating System and the application will be done via the native APIs for each OS. On Android this will consist of the numerous Java APIs that is built into the system, and will likely be used to interact with the camera, file system, and (for sharing) the data connection. On iOS an Objective-C API suite will be used - in keeping with the native development philosophy of iOS.

4.4 Quality Requirements

- Status Messages
  - During graph generation a loading icon will be displayed to show that the system is still busy, given that generation is still progressing normally
  - A message will be displayed when graph generation is complete
  - An error message will be displayed if the graph generation fails for some reason
  - Should the user minimise the application, status messages will be displayed using the notifications pane
- Data extraction from the images and graph generation should take a maximum of 10 seconds
- Generated graphs should be mapped on to the correct surface in the appropriate orientation
- Generated graphs should be scaled correctly and visible on the screen
- For saving & sharing purposes, image size should not exceed 5MB and the resolution should be between 800x600 & 1920x1080
- The camera resolution should not be below 5MP to ensure that accurate OCR analysis is conducted
- The application has to be responsive, that is, the application should react to touch within a second so that no lag is apparent.

4.5 Architecture Constraints

- Android
- iOS

Although no other specific constraints are specified, it is implied that the systems used must all be cross-platform to allow for the two different interfaces (Android and iOS). As such, the AR Engine, OCR Engine and 3D Library must be OS independent.

4.6 Architectural Patterns or Styles

Fundamentally we plan to have our system employ the MVC (Model View Controller), or rather, we employ a derivation of MVC called MVP (Model View Presenter. In the case of Android (the OS we will initially focus on) these MVP segments are as follows:

- **Model -** this is the data access layer - possibly a database API, remote server API or, as in our case, simply the device's file system API.

- **View -** this is the layer that displays information to the user and reacts to user input. On Android, this may be the Activity Class or a Dialog.

- **Presenter -** the Presenter handles the background tasks such as sending and receiving data to and from the Model and View. It also handles other background tasks.

  MVP separates the system into the above mentioned blocks in order to make them less dependent on one another and on most lifecycle-related events. Other advantages are discussed below.

  We have decided to use this architectural pattern due to its pluggability and maintainability. By separating the system into these basic pieces we make troubleshooting easier as well as making the problem solving simpler (one need only focus on one layer at a time). We hope to also simplify the implementation of the system somewhat through this division of complex tasks into smaller - more manageable - tasks.

4.7 Technologies

The application has 3 basic functions:

- Data Gathering - through OCR and user input using Tesseract for OCR.

- Graph Generation - by using Unity 3D/OpenGL.

- Augmented Reality - we will use Vuforia AR SDK to project the 3D graph onto the image marker.

Finally we will be developing the application for Android initially, but we will be porting the app to iOS in future. To accommodate this, all the technologies listed here have APIs available to both operating systems.

5 Functional requirements and application design

5.1 Use case prioritization

5.1.1 Critical

- Generate Graph
- Display Graph

5.1.2 Important

- Editing Graphs
- Save Graph

5.1.3 Nice to Have

- View Previous Graphs
- Share Graph

5.2 Use case/Services contracts

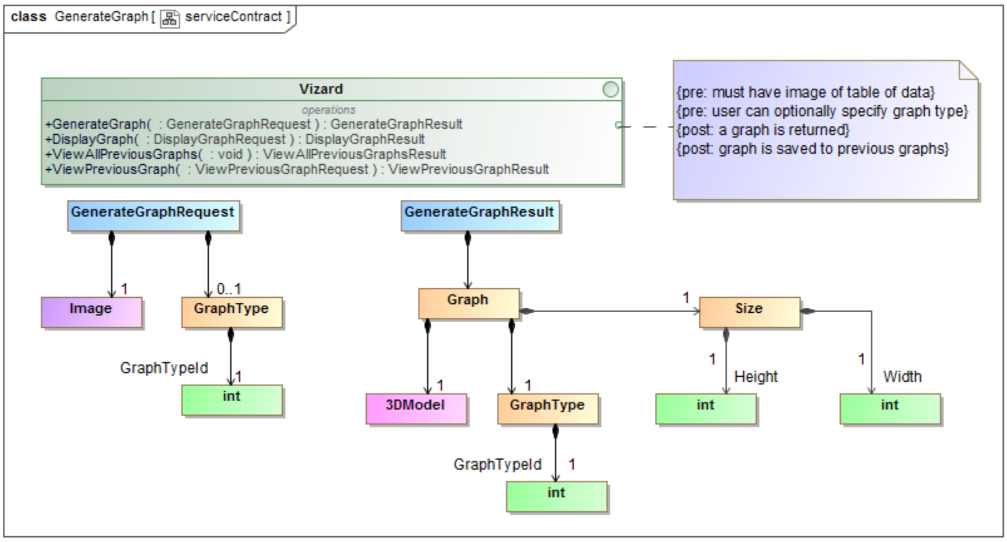

Figure 1: Services Contract : GenerateGraph

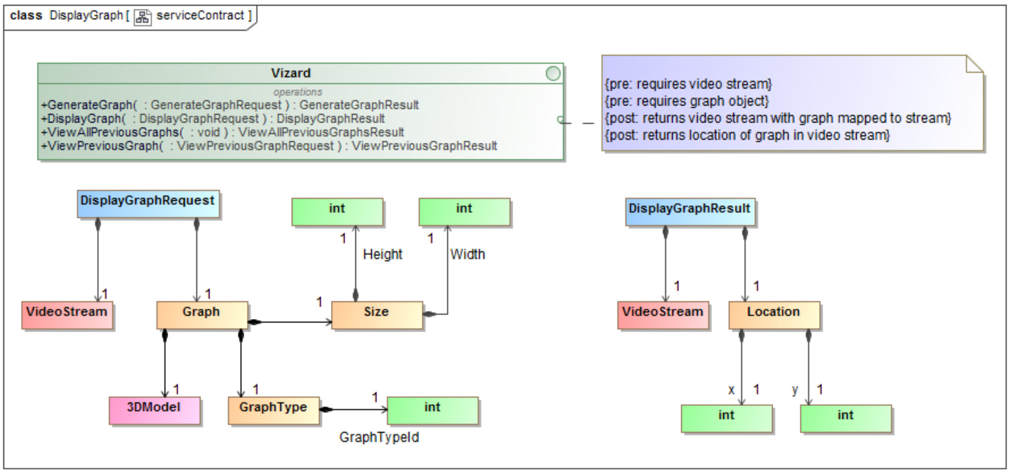

Figure 2: Services Contract : DisplayGraph

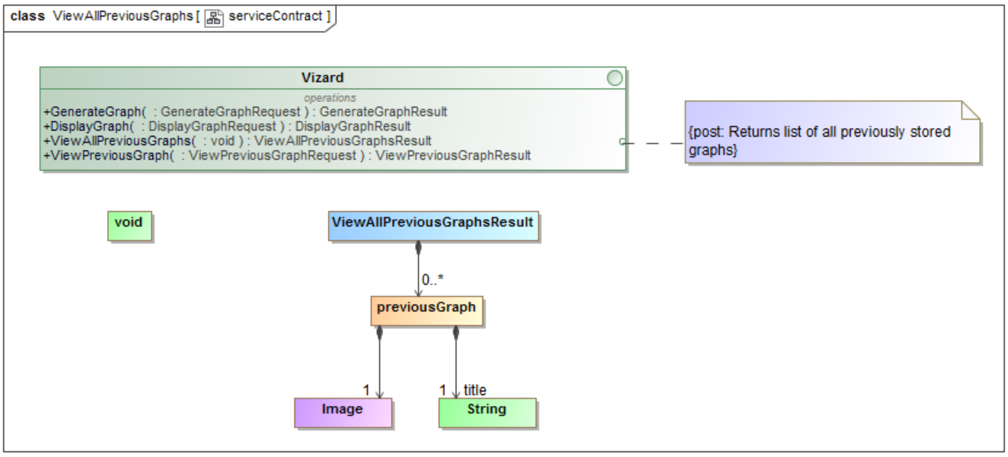

Figure 3: Services Contract : ViewAllPreviousGraphs

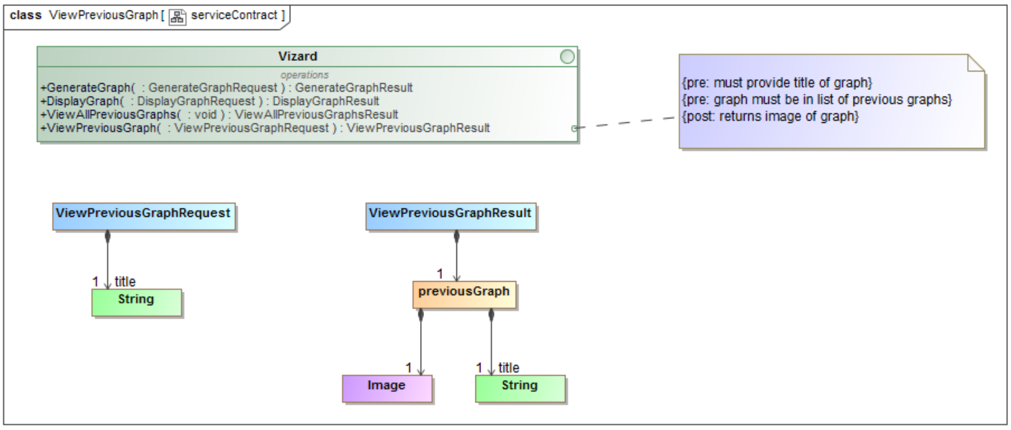

Figure 4: Services Contract : ViewPreviousGraph

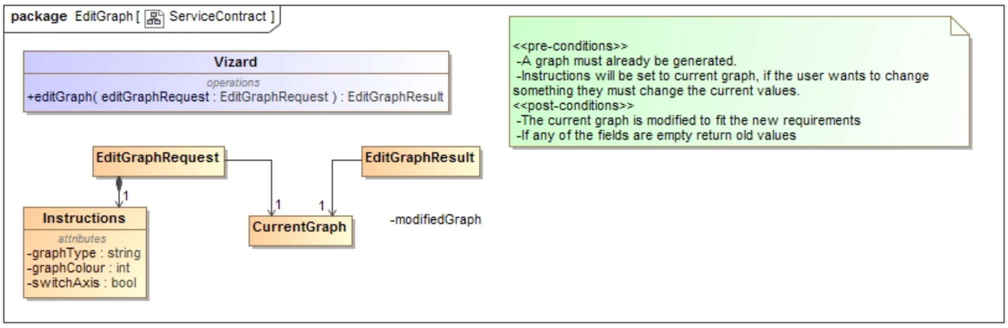

Figure 5: Services Contract : EditGraph

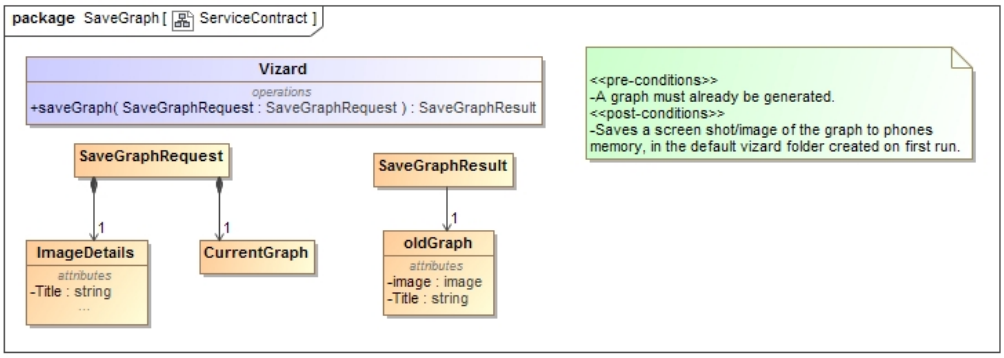

Figure 6: Services Contract : SaveGraph

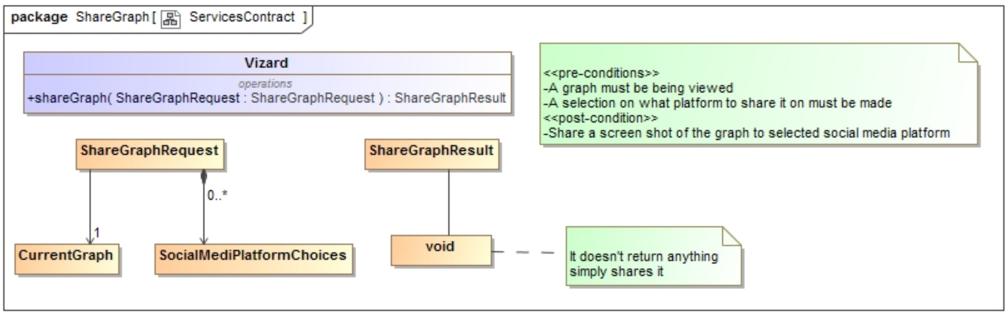

Figure 7: Services Contract : ShareGraph

5.3 Required functionality

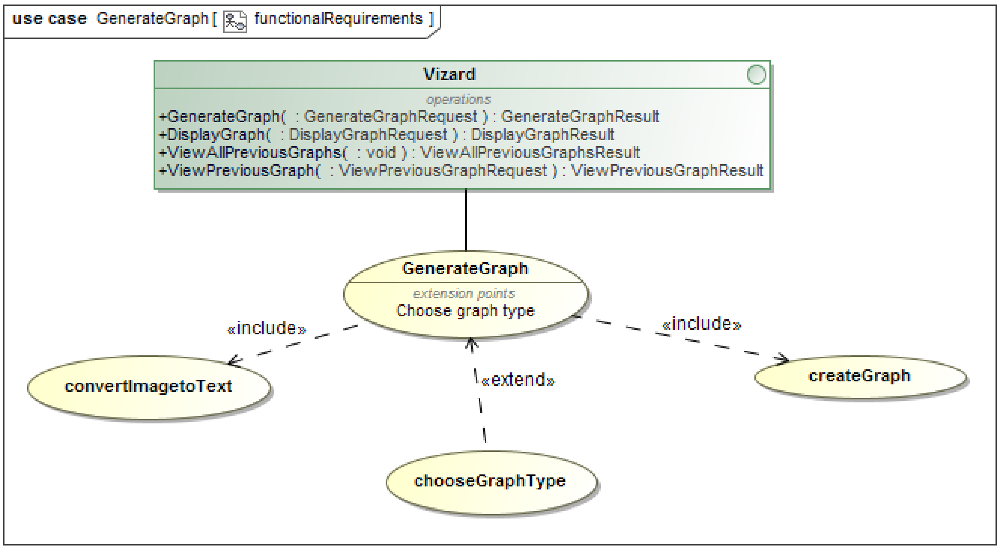

Figure 8: Required functionality : GenerateGraph

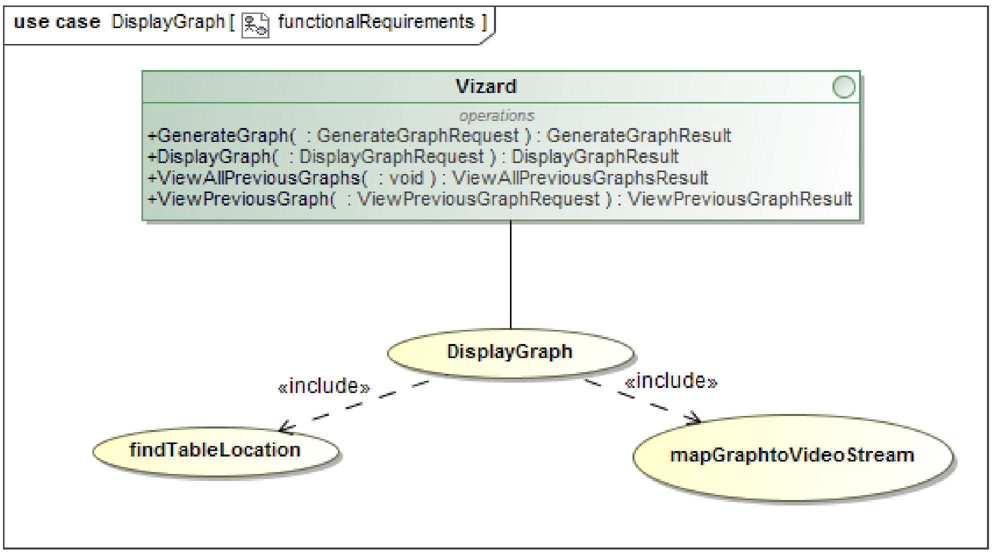

Figure 9: Required functionality : DisplayGraph

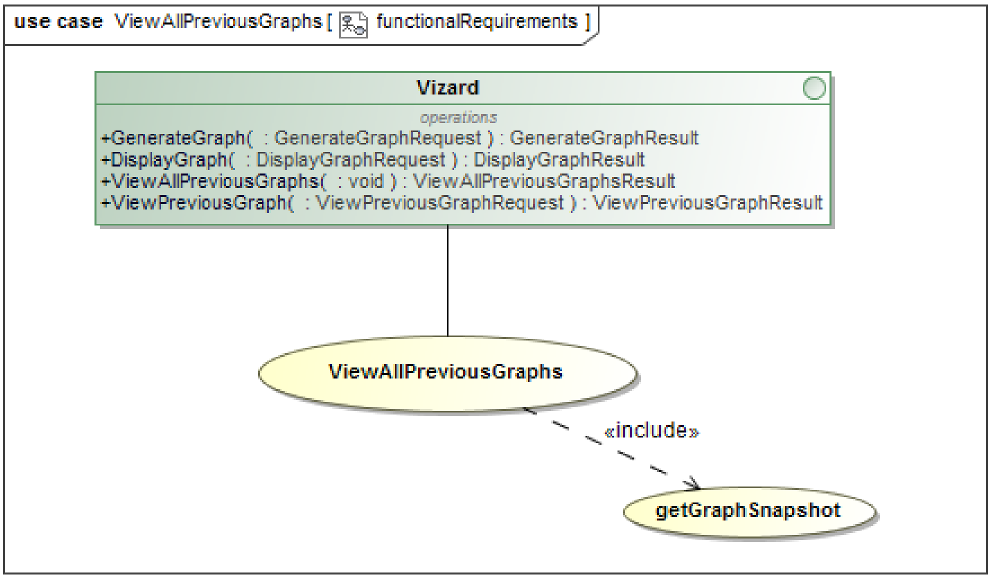

Figure 10: Required functionality : ViewAllPreviousGraphs

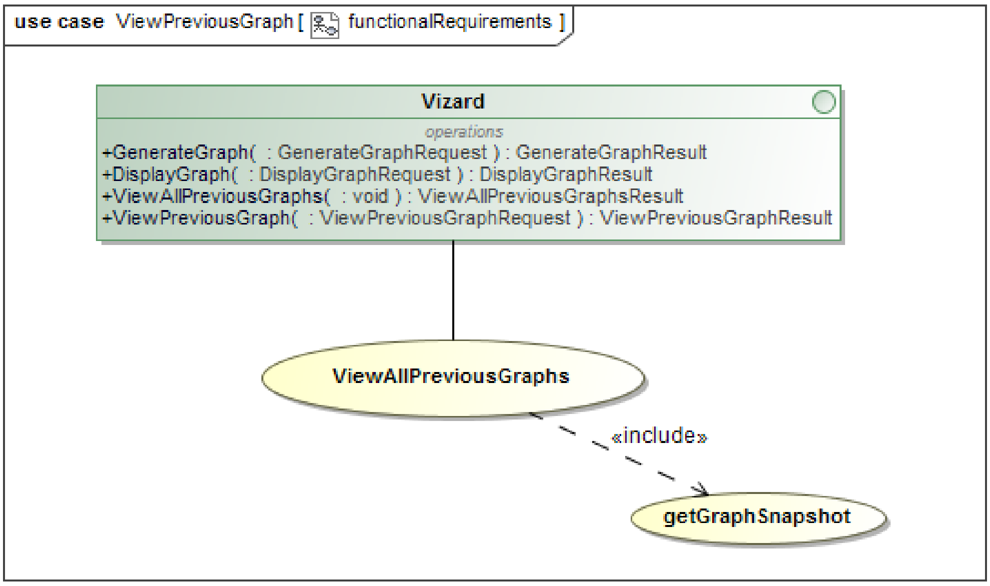

Figure 11: Required functionality : ViewPreviousGraph

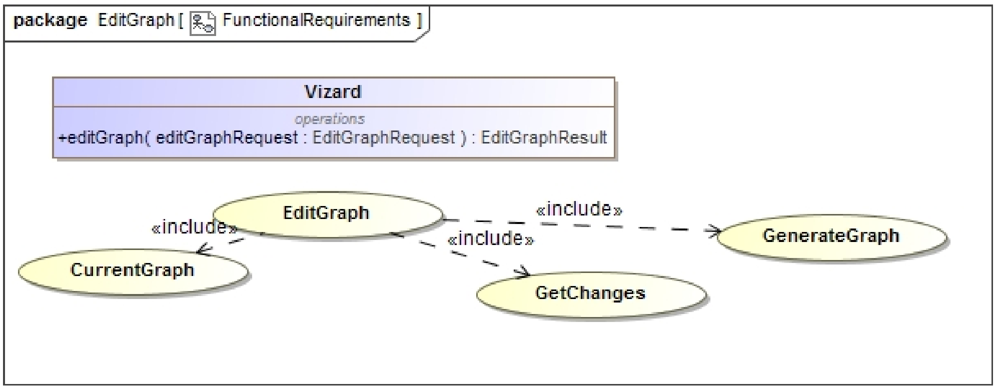

Figure 12: Required functionality : EditGraph

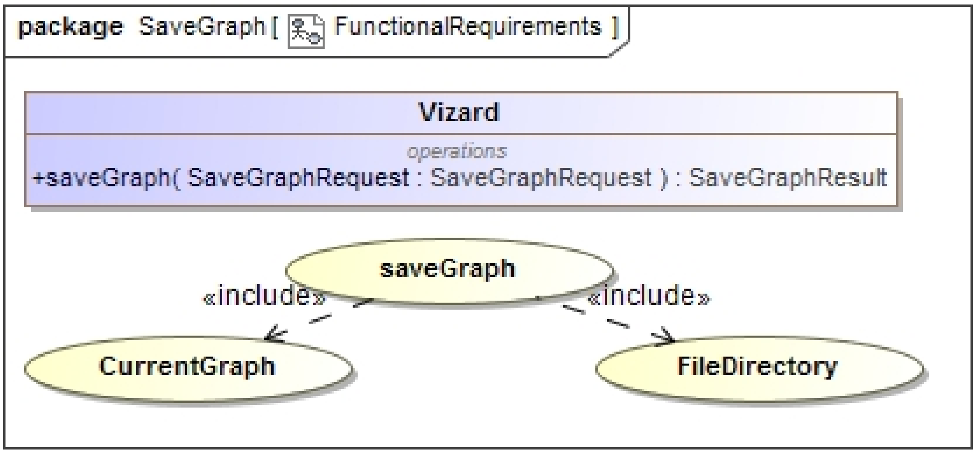

Figure 13: Required functionality : SaveGraph

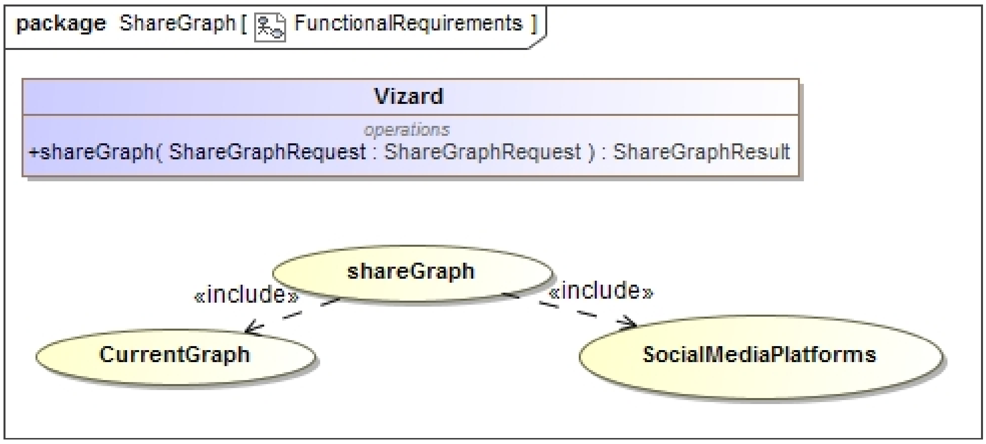

Figure 14: Required functionality : ShareGraph

5.4 Process specifications

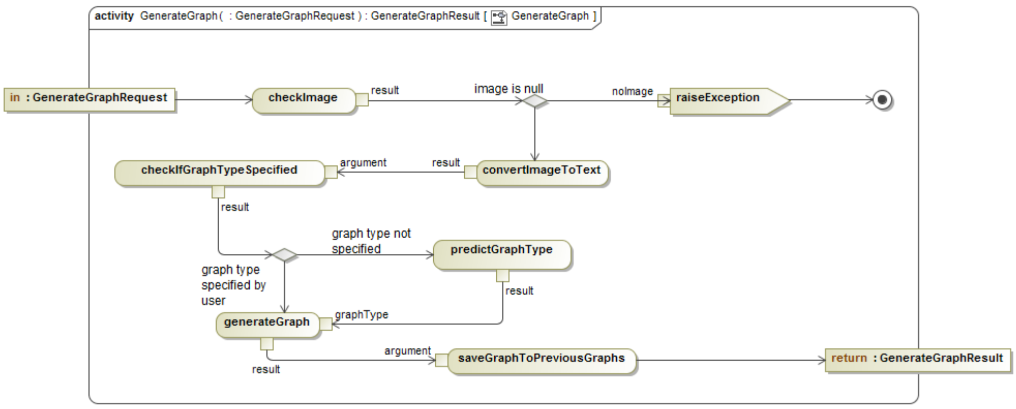

Figure 15: Process specifications : GenerateGraph

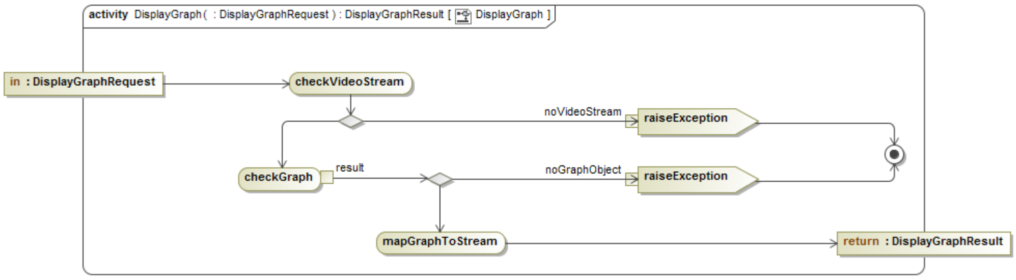

Figure 16: Process specifications : DisplayGraph

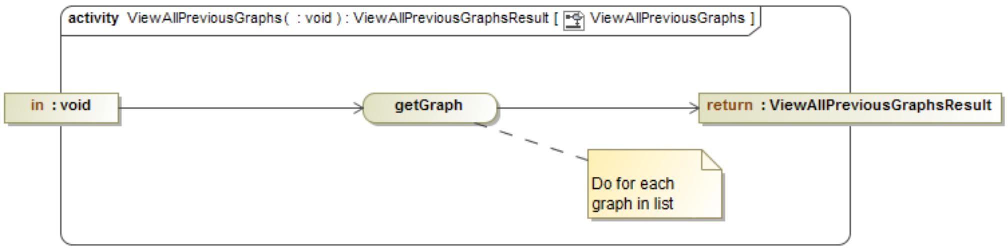

Figure 17: Process specifications : ViewAllPreviousGraphs

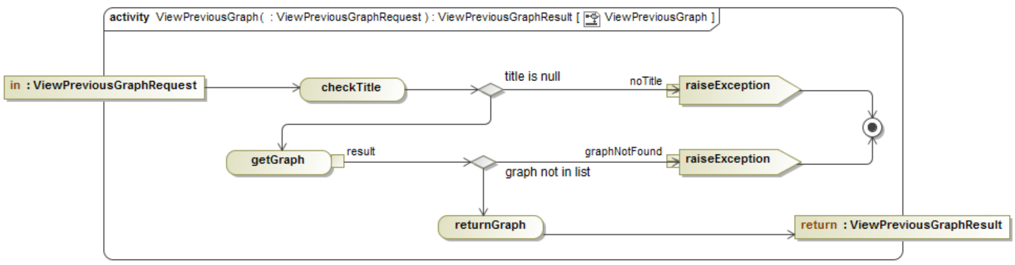

Figure 18: Process specifications : ViewPreviousGraph

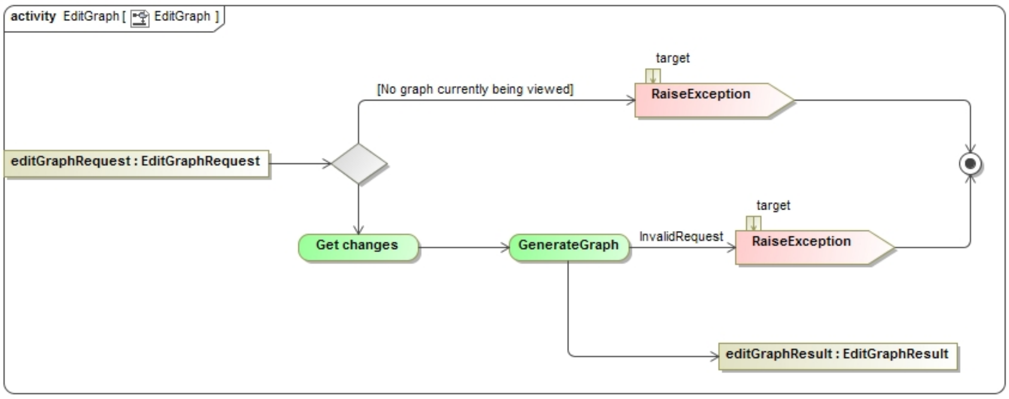

Figure 19: Process specifications : EditGraph

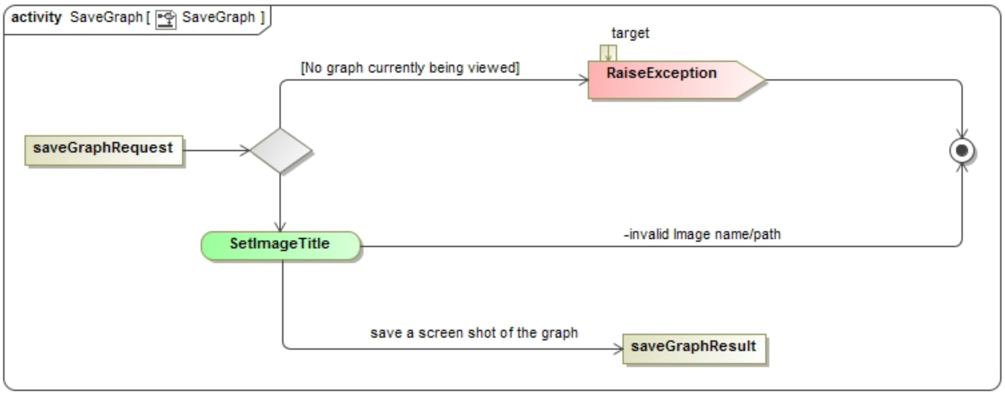

Figure 20: Process specifications : SaveGraph

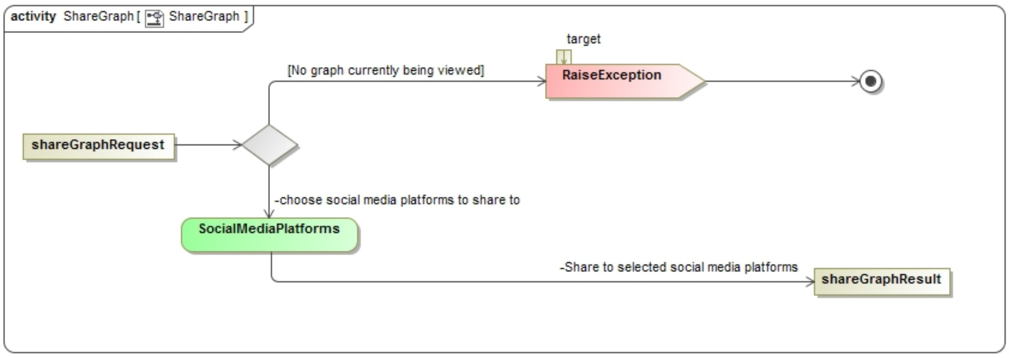

Figure 21: Process specifications : ShareGraph

6 Open Issues