UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS 301 Main Project

Master Specification

ThinkTech

Group Members:

Goodness Adegbenro 13046412
Tshepiso Magagula 12274195
Hlavutelo Maluleke 12318109
Xoliswa Ntshingila 13410378
Lelethu Zazaza 13028023

Git repository link:

`https://github.com/COS301-ThinkTech/Flowchart-planning-and-simulation-tool`

Version 0.1

October 29, 2015

Contents

1 Background, Vision and Scope

1.1 Project background

Without the correct or adequate number of resources, learning a new concept may turn out to be a daunting task. This is especially true for students who are studying a Computer Science course yet are not equipped with a practical or theoretical programming background. The motivation behind this project is to develop a tool that is intended to bridge the gap between inexperience and practical application. The tool will provide the means to simulate program logic through flowcharts in a practical setting.

1.2 Project vision

The aim of this project is to develop a flowchart and planning simulation tool that is simple and intuitive to use. Firstly, this will be accomplished by enhancing the visual nature of the application by making tools in certain contexts more prominent. Additionally, the application should provide informative and clear feedback to the user during the flowchart development phase. It is important that the look-and-feel of the application is uncluttered and uncomplicated so that the user feels at ease to experiment with the tools thereby enhancing the learning experience.

1.3 Project scope

The application is compromised of 2 units: flowchart development and flowchart simulation. An explanation of each unit will follow below.

- Flowchart development

  The user is presented with a canvas upon which he can drop flowchart components to create a complete flowchart. The application will perform error checking on the constructed flowchart, so that (for example) multiple entry points into a program or certain flowchart components are not allowed. The applicatiom should also warn the user of instances of infinite loops or other logical error possibilities.

- Flowchart simulation

  The user should be able to run an error-free flowchart from start to finish. The system should allow for one-click execution of the entire program, as well as step-by-step execution. At all stages during execution, the currently executing component should be highlighted, as well as the connection path being followed. The program's execution should be very visually apparent and appealing. The output of the flowchart's execution should also be apparent.

The following components are specifically excluded from the scope of the project:

- No executable program code generation will be required for this project.
- No complex design elements (such as user-defined component assemblies) are required. Only the basic components of standard flowcharts are necessary.

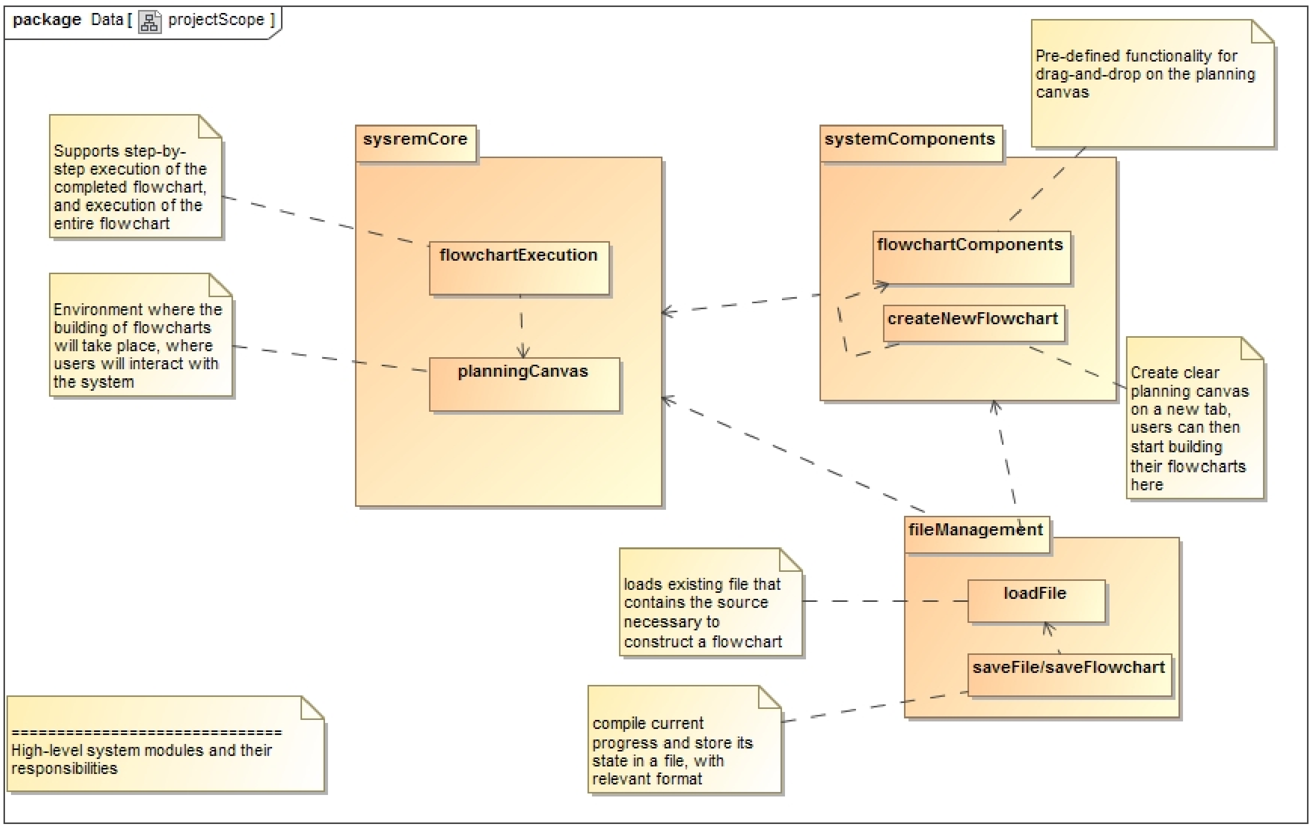

package Data [projectScope]
Pre-defined functionality for drag-and-drop on the planning canvas
Supports step-by-step execution of the completed flowchart, and execution of the entire flowchart
sysremCore
flowchartExecution
planningCanvas
Environment where the building of flowcharts will take place, where users will interact with the system
systemComponents
flowchartComponents
createNewFlowchart
Create clear planning canvas on a new tab, users can then start building their flowcharts here
fileManagement
loadFile
saveFile/saveFlowchart
loads existing file that contains the source necessary to construct a flowchart
compile current progress and store its state in a file, with relevant format
==============================
High-level system modules and their responsibilities

Figure 1: High-level system modules and their responsibilities

2 Use case prioritization

Critical	Important	Nice-To-Have
createFlowchartProject	addFlowchartComponent	drag-and-drop components into bin
deleteFlowchart	editFlowchartComponent	snap-to-grid development
	deleteFlowchartComponent	predefined math functions
	saveFlowchart	infinite-loop detection
	loadFlowchart	logical-error detection
	executeFlowchart	look-and-feel modification

Table 1: Use case prioritization

3 Use cases

3.1 createFlowchartProject

Creates an environment to enable users to start building flowcharts.

Pre Condition: Planning canvas must be blank.

Post Condition: New canvas with Start and Return component created.
Post Condition: Flowchart Planning and Simulation tools ready for use.

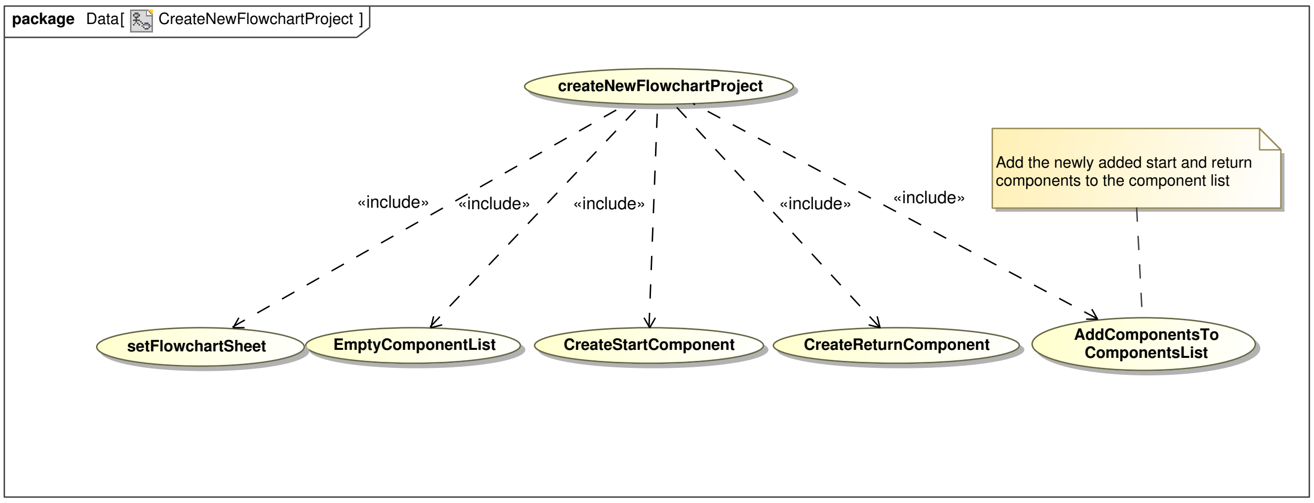

package Data[CreateNewFlowchartProject]
createNewFlowchartProject
«include»
«include»
«include»
«include»
«include»
Add the newly added start and return components to the component list
setFlowchartSheet
EmptyComponentList
CreateStartComponent
CreateReturnComponent
AddComponentsTo ComponentsList

Figure 2: createFlowchartProject Use Case Diagram

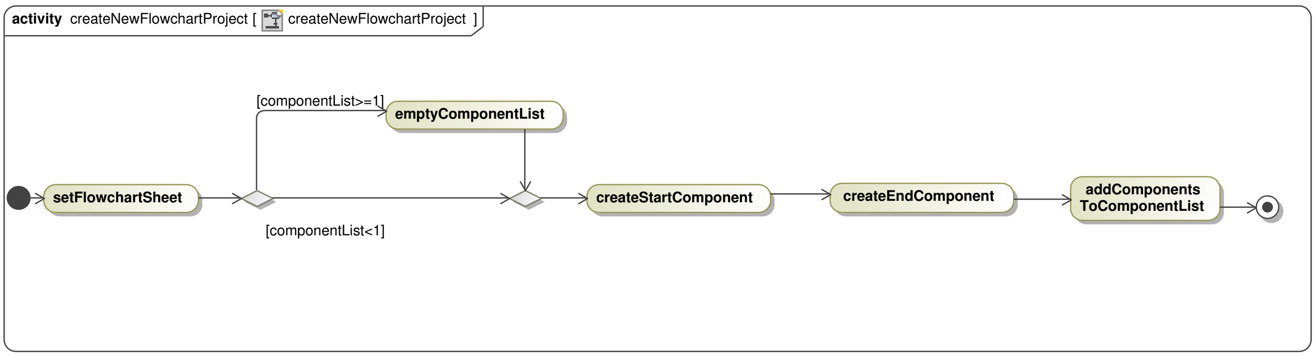


Figure 3: createFlowchartProject Activity Diagram

3.2 addFlowchartComponent

Provides users with functionality to select the flowchart components they wish to place on the planning canvas.

Pre Condition: Canvas is available.

Post Condition: Component has been added to flowchart and appers on canvas with the necessary connections, if any.

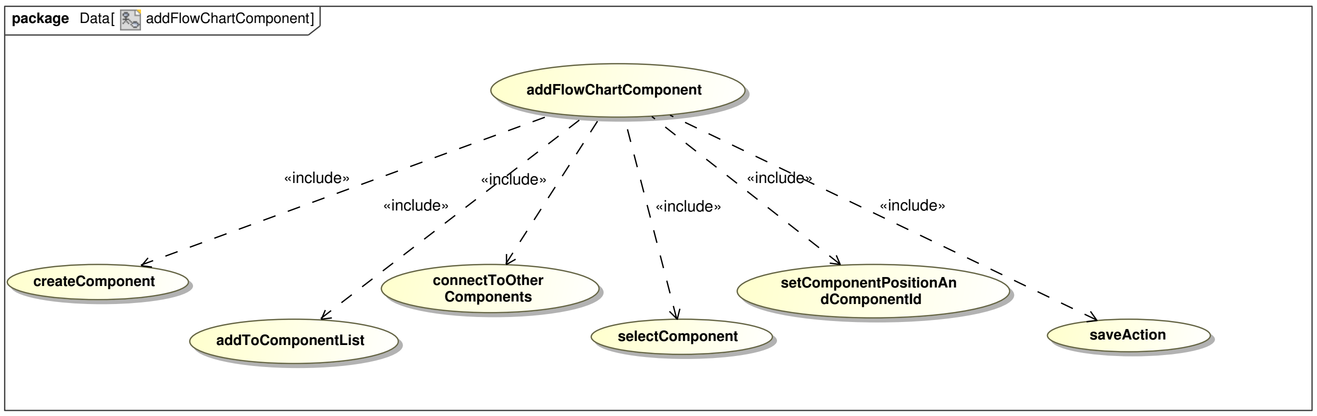


Figure 4: addFlowchartComponent Use Case Diagram

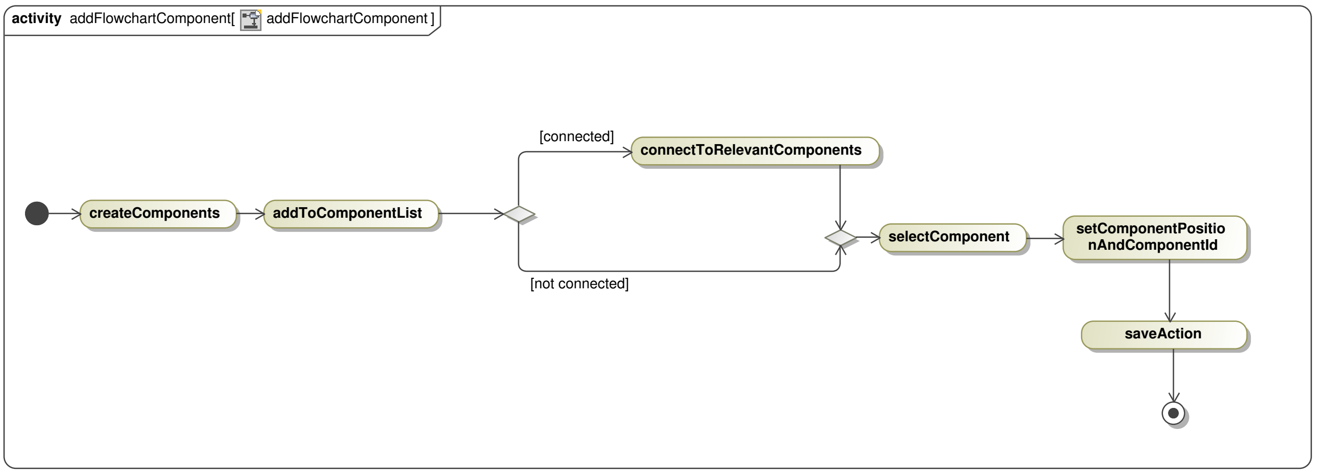

activity addFlowchartComponent[addFlowchartComponent]
createComponents
addToComponentList
[connected]
connectToRelevantComponents
[not connected]
selectComponent
setComponentPositio nAndComponentId
saveAction

Figure 5: addFlowchartComponent Activity Diagram

3.3 editFlowchartComponent

The editFlowchartComponent use case provides functionality for the user to edit each component of the flowchart on the canvas

Pre Condition: Canavas must exist.
Pre Condition: Component must exist on active canvas.

Post Condition: Component has been edited.
Post Condition: Edited component has been saved.

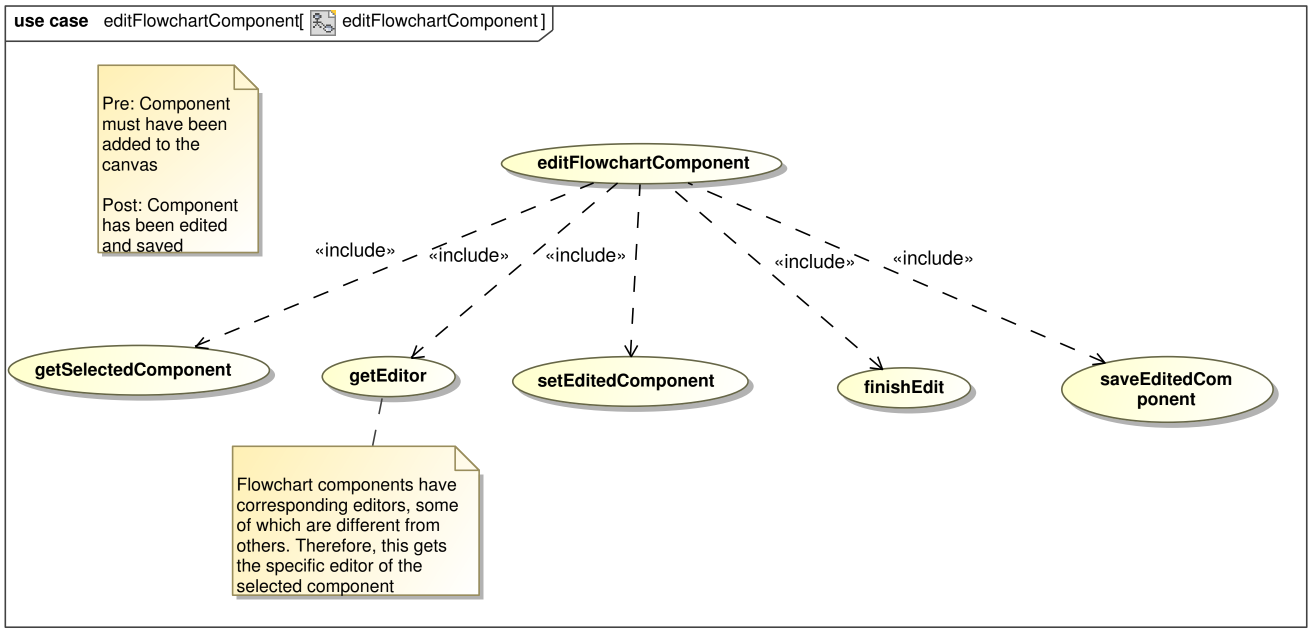

use case editFlowchartComponent[editFlowchartComponent]
Pre: Component must have been added to the canvas
Post: Component has been edited and saved
editFlowchartComponent
«include»
«include»
«include»
«include»
«include»
getSelectedComponent
getEditor
setEditedComponent
finishEdit
saveEditedComponent
Flowchart components have corresponding editors, some of which are different from others. Therefore, this gets the specific editor of the selected component

Figure 6: editFlowchartComponent Use Case Diagram

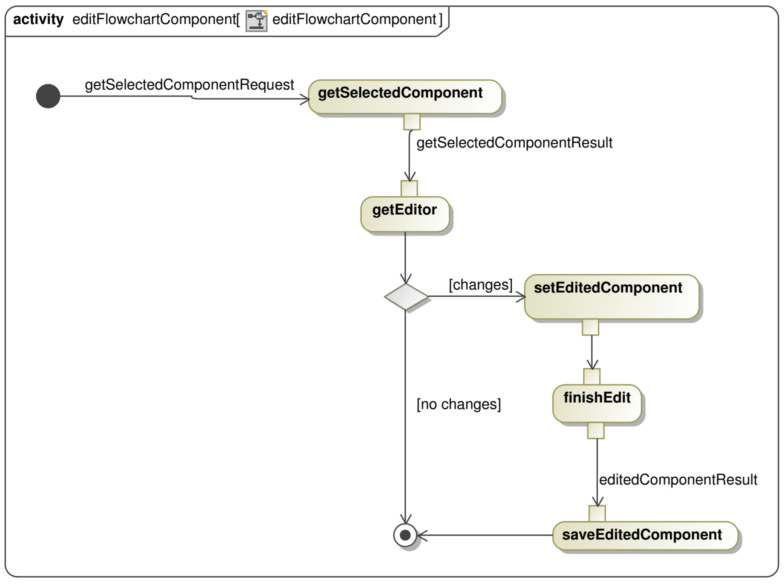


Figure 7: editFlowchartComponent Activity Diagram

3.4 deleteFlowchartComponent

The deleteFlowchartComponent use case enables the functionality to delete individual components from the canvas.

Pre Condition: The canvas has to be available.
Pre Condition: Component exists in the canvas space and is in the components list.

Post Condition: The canvas is clear of any components that were selected for removal.

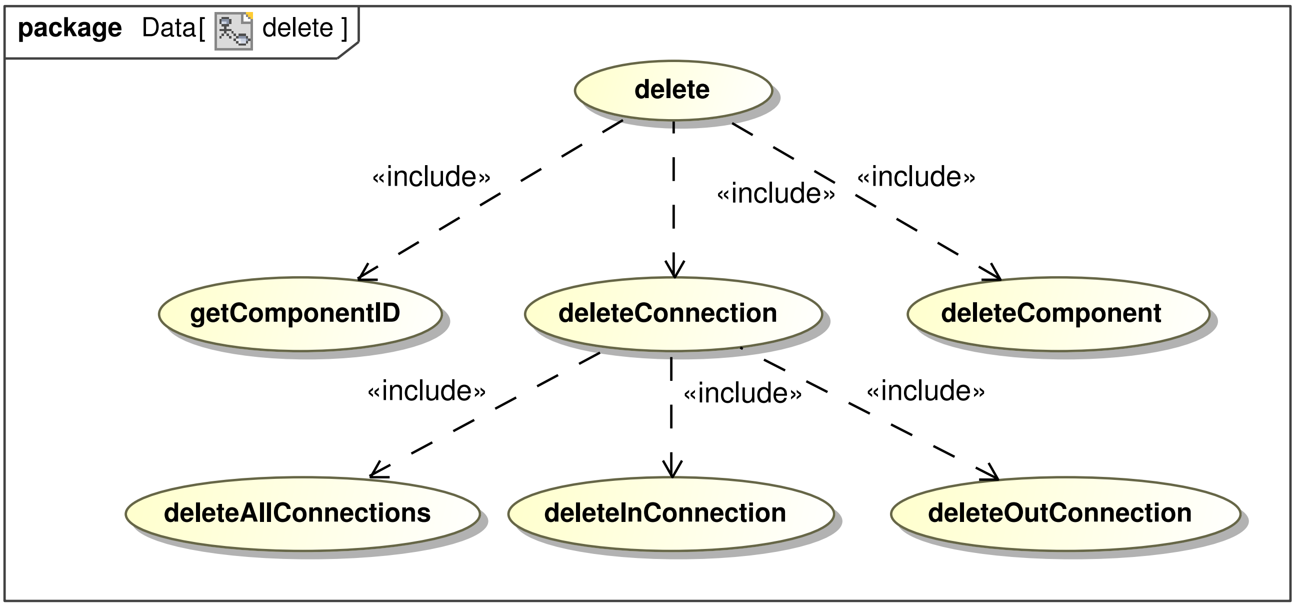


Figure 8: deleteFlowchartComponent Use Case Diagram

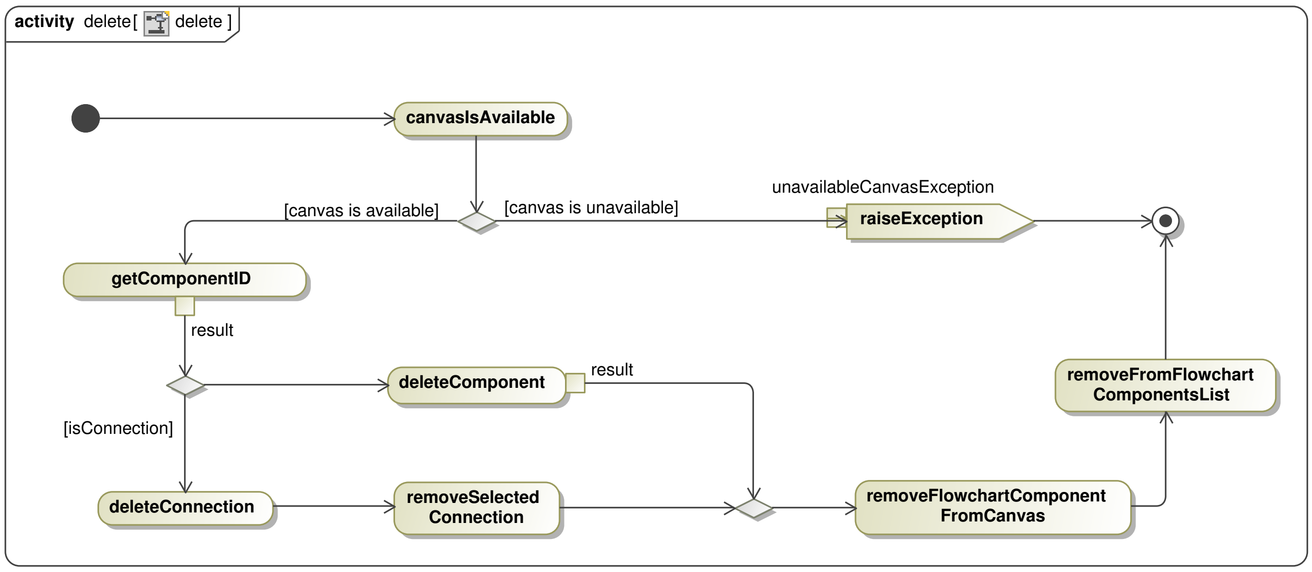


Figure 9: deleteFlowchartComponent Activity Diagram

3.5 deleteFlowchart

The deleteFlowchartProject use case serves the purpose of removing a flowchart project.

Pre Condition: Canvas exists.
Pre Condition: Canvas is active.
Pre Condition: The flowchart to be deleted must exist.
Pre Condition: The flowchart to be deleted must be active.

Post Condition: The flowchart must be removed from the workspace and a new flowchart must then be created and made active on the workspace.

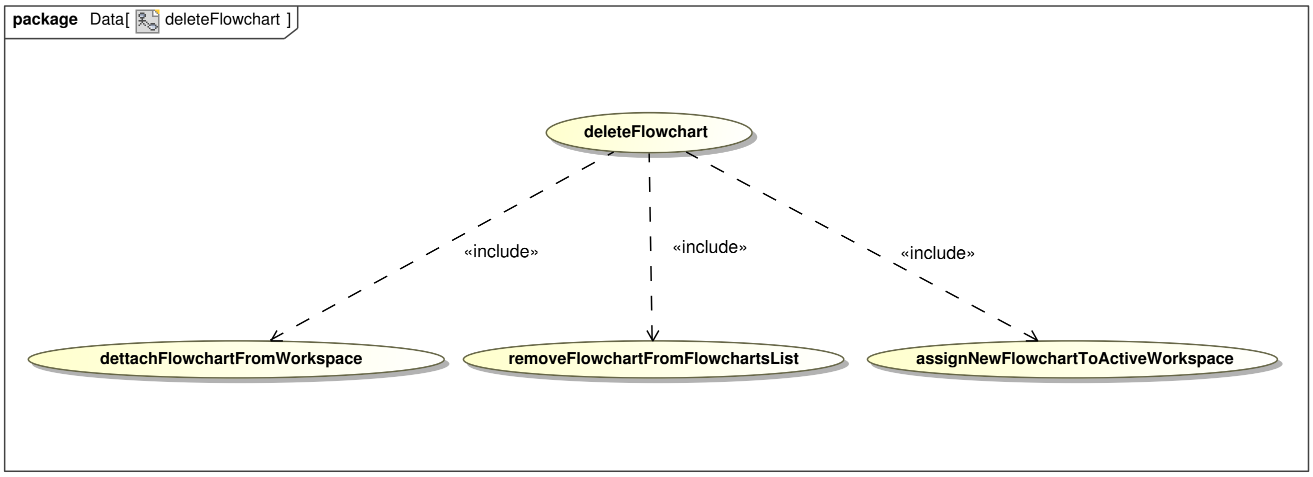

package Data[deleteFlowchart]
deleteFlowchart
«include»
«include»
«include»
dettachFlowchartFromWorkspace
removeFlowchartFromFlowchartsList
assignNewFlowchartToActiveWorkspace

Figure 10: deleteFlowchart Use Case Diagram

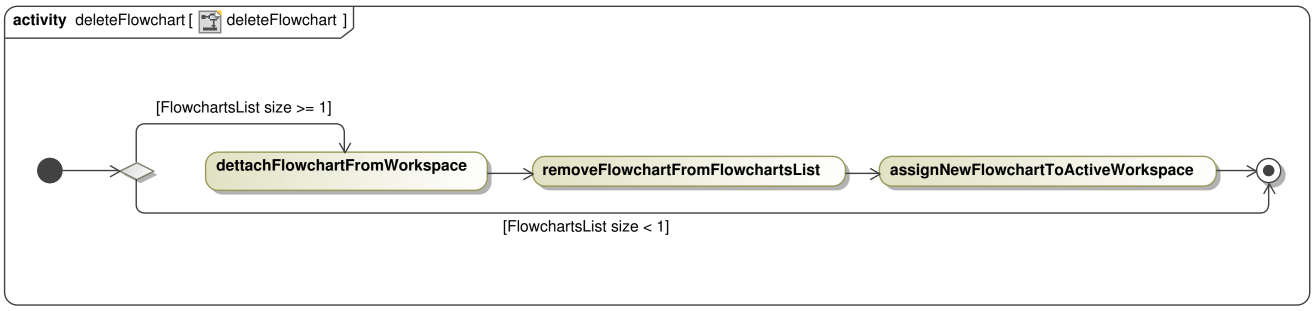

activity deleteFlowchart [deleteFlowchart]
[FlowchartsList size >= 1]
dettachFlowchartFromWorkspace
removeFlowchartFromFlowchartsList
assignNewFlowchartToActiveWorkspace
[FlowchartsList size < 1]

Figure 11: deleteFlowchart Activity Diagram

3.6 saveFlowchart

The saveFlowchart use case provides functionality for the user to save a flowchart.

Pre Condition: Canvas exists.
Pre Condition: Canvas is active.

Post Condition: Flowchart has been saved to file.

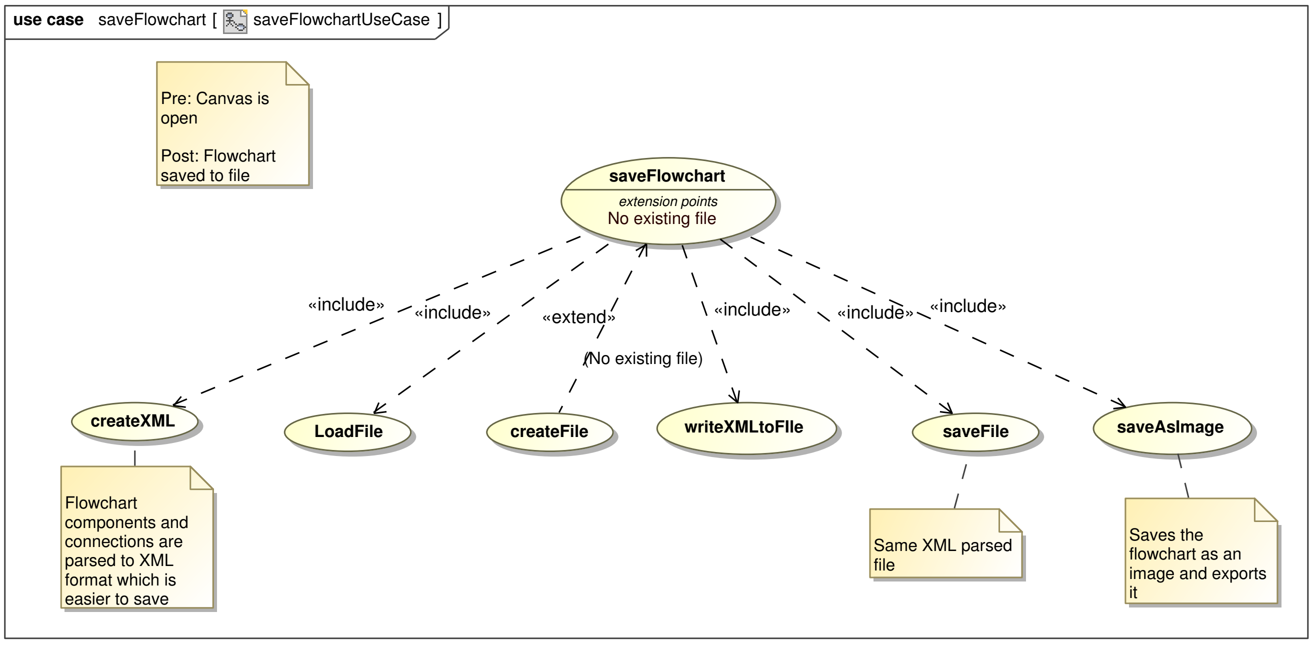


Figure 12: saveFlowchart Use Case Diagram

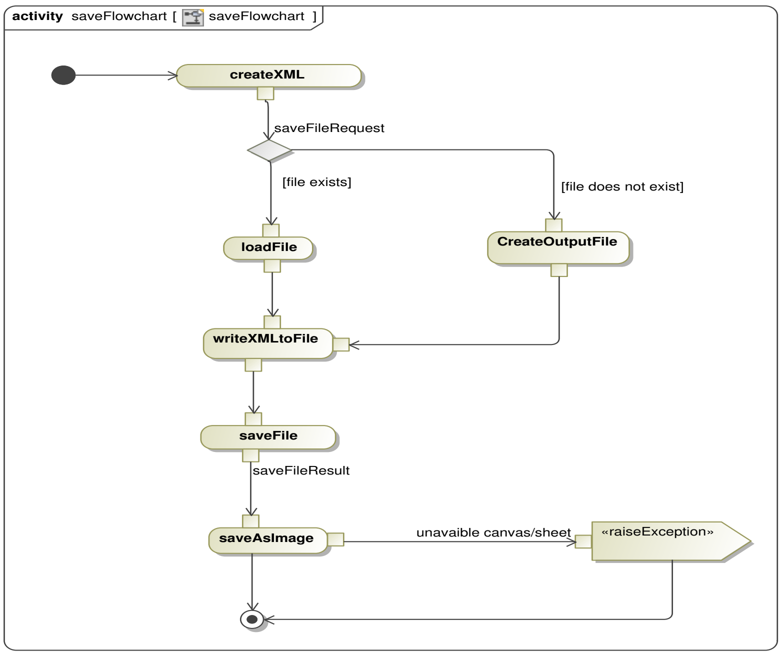


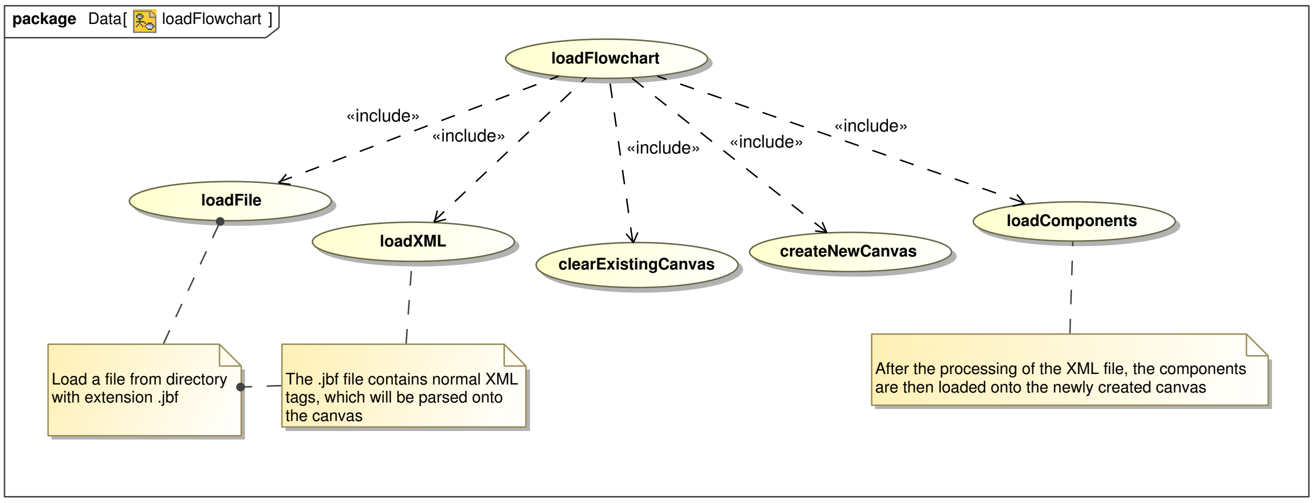
Figure 13: saveFlowchart Activity Diagram

3.7 loadFlowchart

The loadFlowchart use case allows users load flowcharts from existing files. The file is editable and can be modified by the user.

Pre Condition: File with correct extension exists.

Post Condition: File contents have been converted and loaded on to the canvas.



package Data[loadFlowchart]
loadFlowchart
«include»
«include»
«include»
«include»
«include»
loadFile
loadXML
clearExistingCanvas
createNewCanvas
loadComponents
Load a file from directory with extension .jbf
The .jbf file contains normal XML tags, which will be parsed onto the canvas
After the processing of the XML file, the components are then loaded onto the newly created canvas

Figure 14: loadFlowchart Use Case Diagram

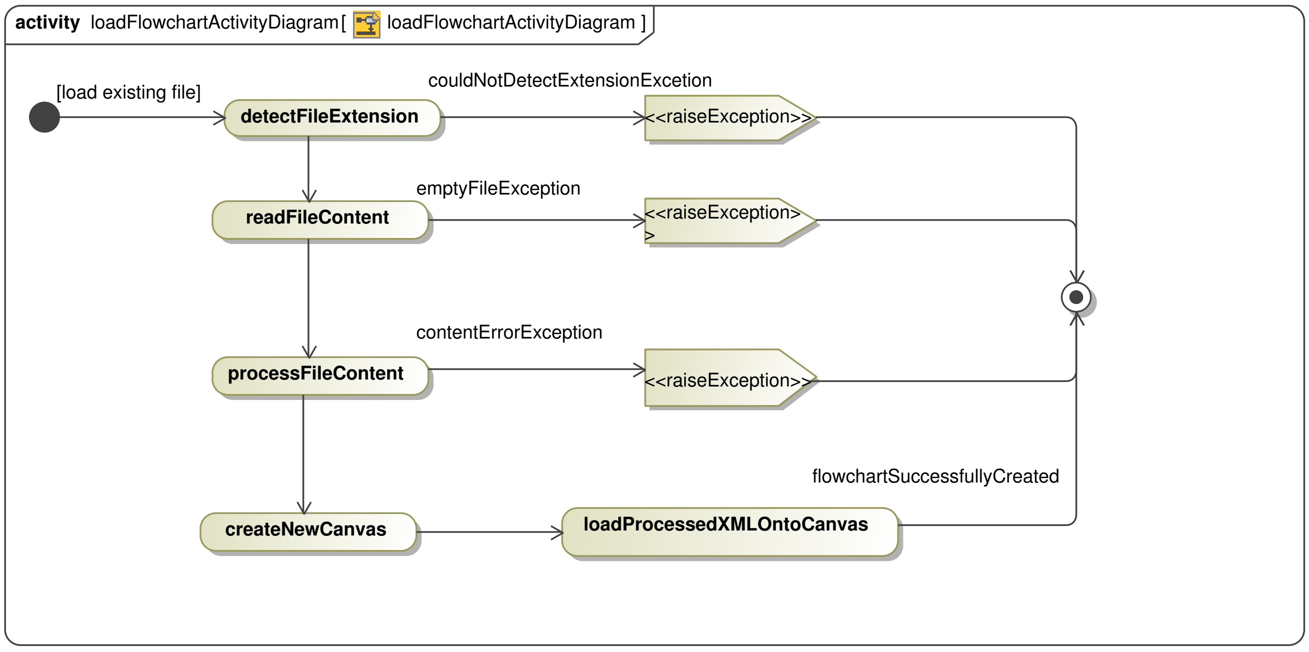


Figure 15: loadFlowchart Activity Diagram

3.8 executeFlowchart

The executeFlowchart use case enables the functionality to execute the flowchart step-by-step or from start-to-end.

Pre Condition: Canvas has to be available.

Post Condition: Flowchart will return feedback of any errors, warnings or successful execution along with the results of any calculations.

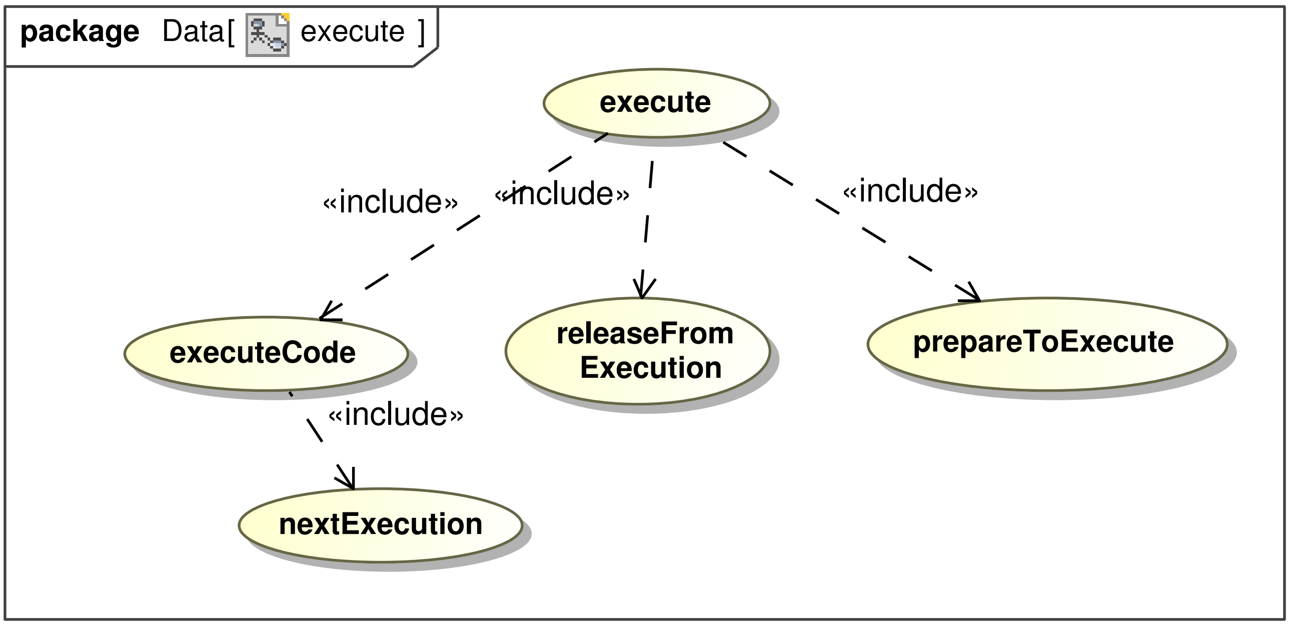


Figure 16: executeFlowchart Use Case Diagram

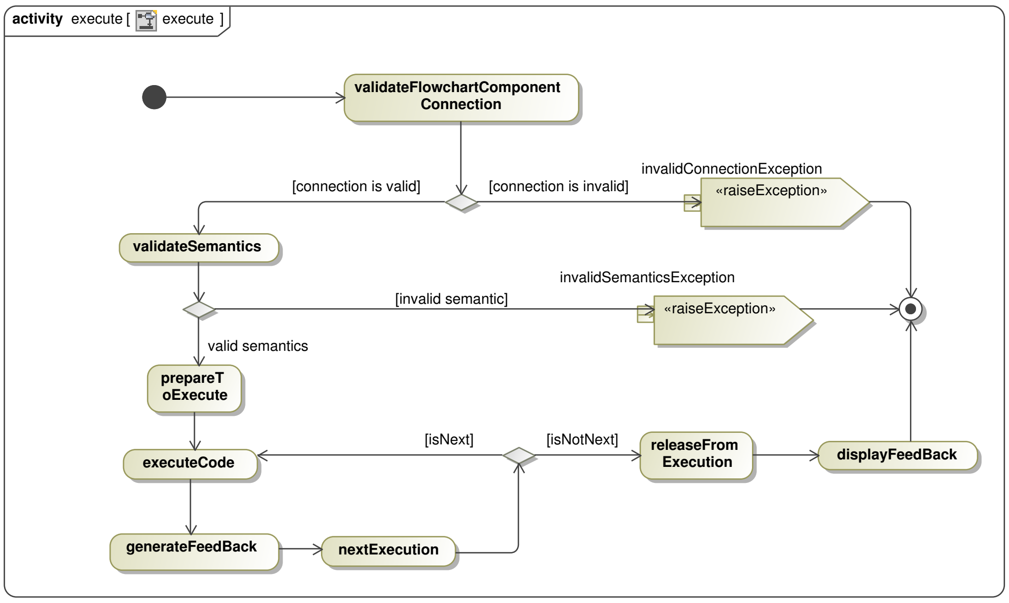


Figure 17: executeFlowchart Activity Diagram

4 The Domain Model - High-level

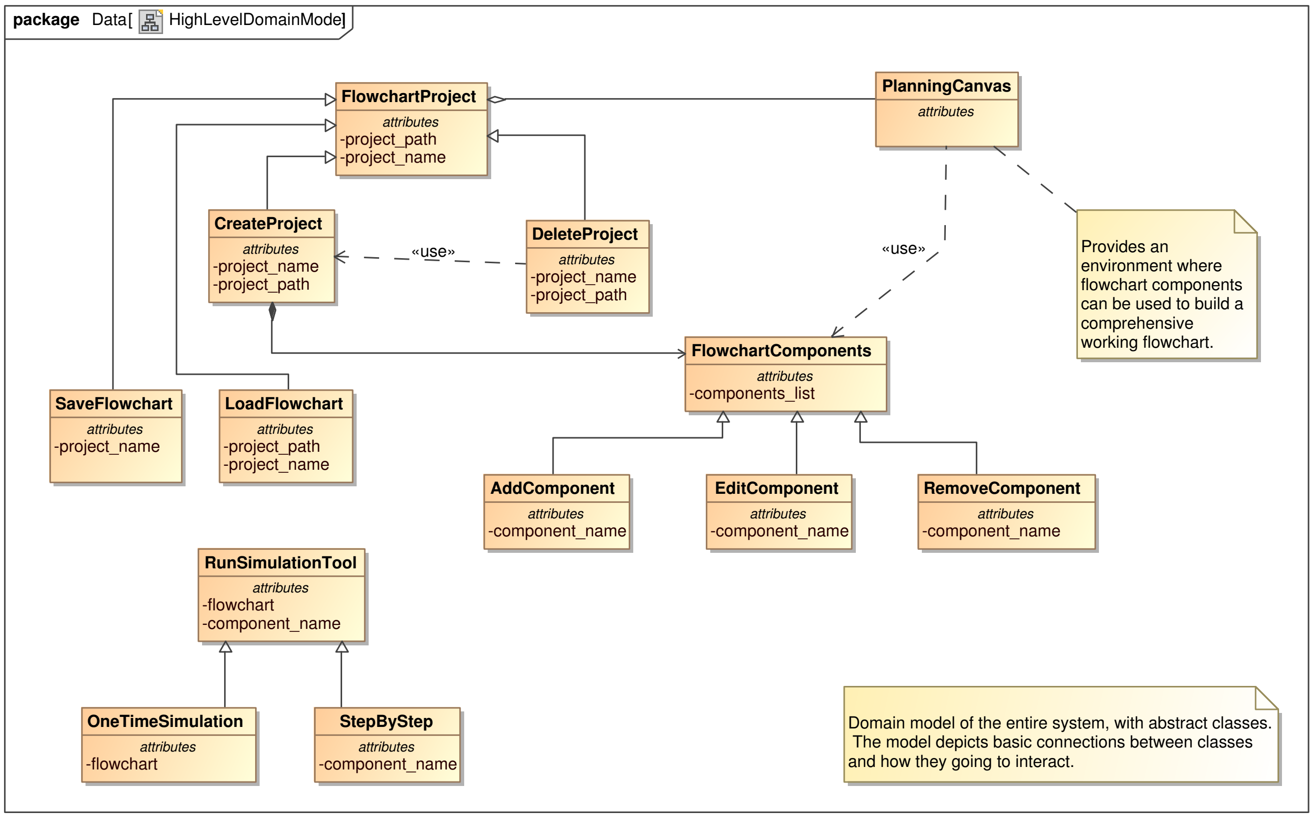


Figure 18: Domain Model Diagram

5 Access and Integration Channels

5.1 Access Channels

5.1.1 Human access channels

This application is accessible on a desktop computer running the Linux operating system, with a possibility of making the application accessible on a desktop computer running the Windows operating system.

5.1.2 System access channels

The system is not required to interface with any existing systems. It is only intended for execution on a desktop computer running the Linux operating system.

5.2 Integration channels

Since no other system interfaces with this system, no integration is required except of that of the internal modules.

6 Architectural Responsibilities

This system does not connect with any network, database or any other systems. Mostly it will depend on pure Java built-in functions.

7 Quality Requirements

Reliability: Any valid program expressed as a flowchart, without errors, is executable. In cases where errors are detected, sufficient feedback will be generated and provided to the user. A flowchart without any errors will generate output and feedback will also be provided.

Performance: Performance is not a major concern. However, to ensure the system must provide feedback to the user during flowchart building and execution in a reasonable amount of time appropriate data structures and adequate design patterns will be used.

Maintainability: Since the whole system will be coded using Object-Oriented Programming (OOP), it will be much easy to update, extend and maintain the system. OOP allows the future development of the system.

Availability: The application is available to all users, and can be executed on Windows and Linux machines.

Security: This is a stand-alone application and it does not connect to any networks, so security is not a real concern.

Testability: The GUI will provide a testing environment to the system, this will provide all the functionalities a standard and a complex flowchart can provide. When thorough GUI testing has been conducted, all detected malfunctions can be corrected.

Usability: With the drag-and-drop functionality, the entire system will be very much easier to use; and also, pre-defined programming operations (eg. loops, conditional statements etc.) will be present. This is solely to enhance users to implement these operations without any standard programming language. Consequently the system will be independent of programming languages.

8 Architecture Constraints

Because this system is built from an already existing system which was purely coded in Java, it is then highly recommended to implement the new system in Java. There are also numerous reasons for this, i.e., Java is a pure Object-oriented programming language which makes the code more readable and easier to read. This enhances maintainability.

And also, the client specified that he needs an Object-oriented application, Java provides more precise Object-oriented features.

9 Technologies

As previously stated that the system that is being upgraded was coded in Java, this system as well will be coded in Java. For testing, JUnit will be used.

To save the current state of a flowchart, all the components of the flowchart will be processed into an XML file, to open the existing file, an XML will be parsed. So, XML will play a central role in managing the files.

10 Architecture design

10.1 Overview

The Model-View-Controller allows for dividing the application in three components, making the problem domain independent from the user interface. A controller as the ability to send commands to the model to update the model's state (e.g., adding a component to the flowchart). It can also send commands to the associated view to change the view's presentation of the model (e.g. maximizing the flowchart window) A model stores data that is retrieved by the controller and displayed in the view. A view requests information from the model that it uses to generate an output representation to the user.

10.2 Architectural tactics addressing quality requirements

This section discusses the architectural tactics which are used to concretely address the quality requirements for the Flowchart and Simulation Tool application.

10.2.1 Contracts based development

Contracts allow the application to adhere to the following quality requirements: Testability and maintainability. The contracts will be enforced fo services with pre and post-conditions which are assessed in unit tests, and data structure constraints enforced through data structure validation.

10.2.2 Dependency Injection

Dependency injection allows for the application to adhere to the following quality requirements: Flexibility, deployability, testability.

10.2.3 Minimize technology suite

In order to improve maintainability the software architecture will minimize the number of programming languages used.

10.3 Architectural components

This section discusses the architectural components and technologies used to address the architectural responsibilities and the architectural tactics chosen to address the quality requirements

10.3.1 Java

Java is chosen as a single programming language used for the application in order to implement the tactic of minimizing the technology suite. Using a single programming language reduces complexity and improves maintainability.

10.3.2 JUnit

JUnit will be used to achieve testability. With unit testing, we ensure that all module unit are precise and produce the expected results.

10.4 Development architecture

The development architecture is the architecture designed to support qualities in the development process itself. This includes qualities like reliability.

10.4.1 Version control

The developer of the project make use of a git repository. Any new feature or bug should be developed in a new branch which is only merged into the trunk once the feature or passes its unit tests.

10.5 IDE

Each developer has the freedom to choose whichever IDE they prefer.

10.6 Builds

To build the project artifacts Ant will be used. Ant is integrated with Netbeans IDE.

10.7 Unit testing

Unit testing will be done using JUnit which is a basic testing environment for Java applications.

10.8 Integration testing

This is a stand

10.9 Bug tracking

The projects will use GitHub's issue tracker for bug tracking.