UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

COS 301 Main Project

Requirements and Design Specifications

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

September 25, 2015

Contents

1 Vision and scope

1.1 Project background

The flowchart planning and simulation tool is a visual tool which aims to introduce programming logic to students by providing a platform to construct and execute flowcharts which depict basic programming structures whilst validating whether the flowchart components are valid .

1.2 Project vision

Most first year computer science students come from a backgroud void of programming to a programming backgroud and they struggle to grasp the logical programming concepts quickly and easily. Therefore, this project sets out to create an application with a simplified and easy-to-use interface for the planning and simulation of flowcharts, which can be used in an education setting for an introductory program logic course. The objective is to develop an application that can feasibly be used in such a practical setting. The application must be visual in nature, due to the visual nature of flowcharts. The application must facilitate exploration and experimentation, while providing clear feedback to the user as to how the program executes.

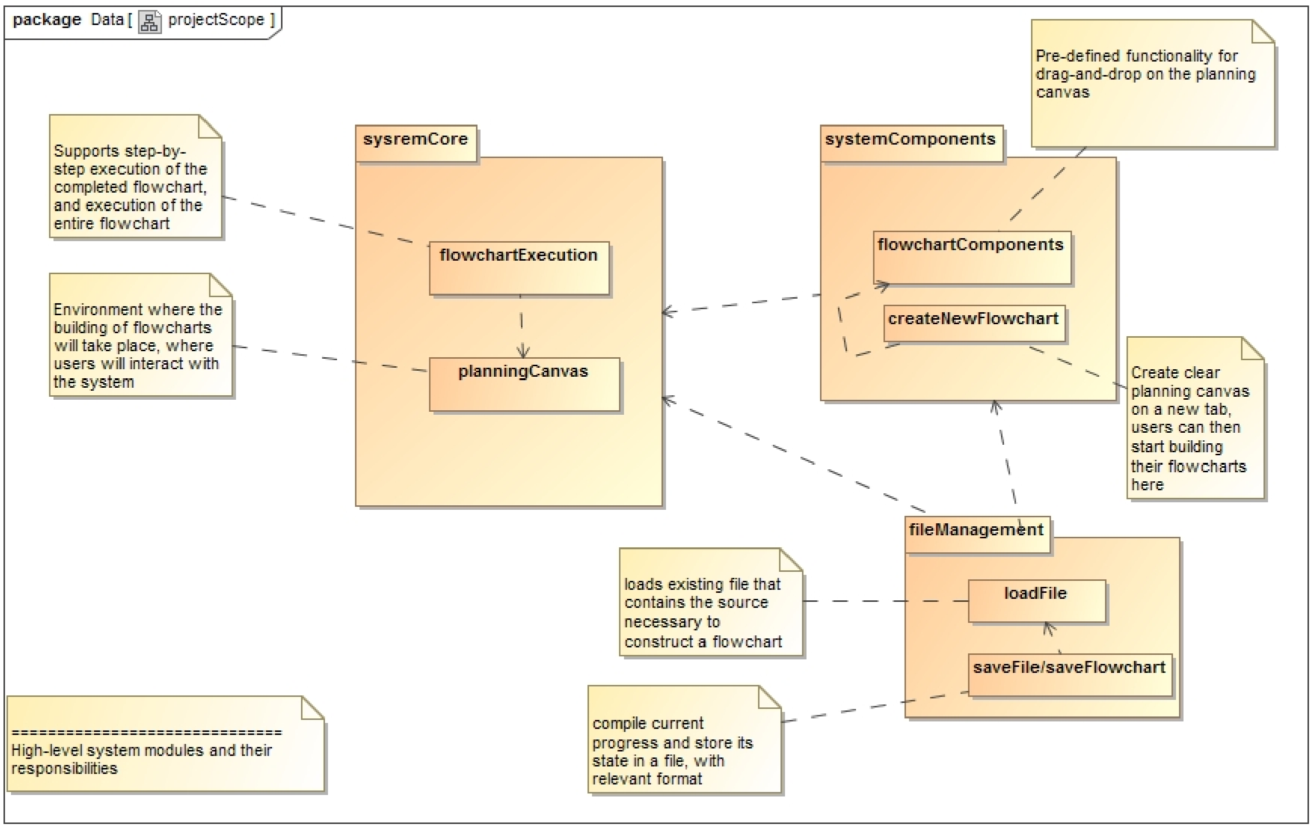
1.3 Project scope

The project consists of the following two components:

- A planning canvas, in which a flowchart can be built using an intuitive drag-and-drop editor. Flowchart components will be selected and dropped into the canvas, and connected into complete flowcharts. The system will also have to do error checking on the constructed flowcharts, so that (for example) multiple entry points into a program or certain flowchart components are not allowed.
- An execution system, in which a flowchart can be run 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]
sysremCore
flowchartExecution
planningCanvas
systemComponents
flowchartComponents
createNewFlowchart
fileManagement
loadFile
saveFile/saveFlowchart
Supports step-by-step execution of the completed flowchart, and execution of the entire flowchart
Environment where the building of flowcharts will take place, where users will interact with the system
Pre-defined functionality for drag-and-drop on the planning canvas
Create clear planning canvas on a new tab, users can then start building their flowcharts here
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

High level system modules and their responsibilities.

2 Use case prioritization

Table 1: Use case prioritization

Critical	Important	Nice-To-Have
createFlowchartProject	addFlowchartComponent	drag-and-drop flowchart into bin
deleteFlowchart	editFlowchartComponent	enchanceExecutionWithColours
	deleteFlowchartComponent	
	saveFlowchart	
	loadFlowchart	
	executeFlowchart	

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 tools ready for use

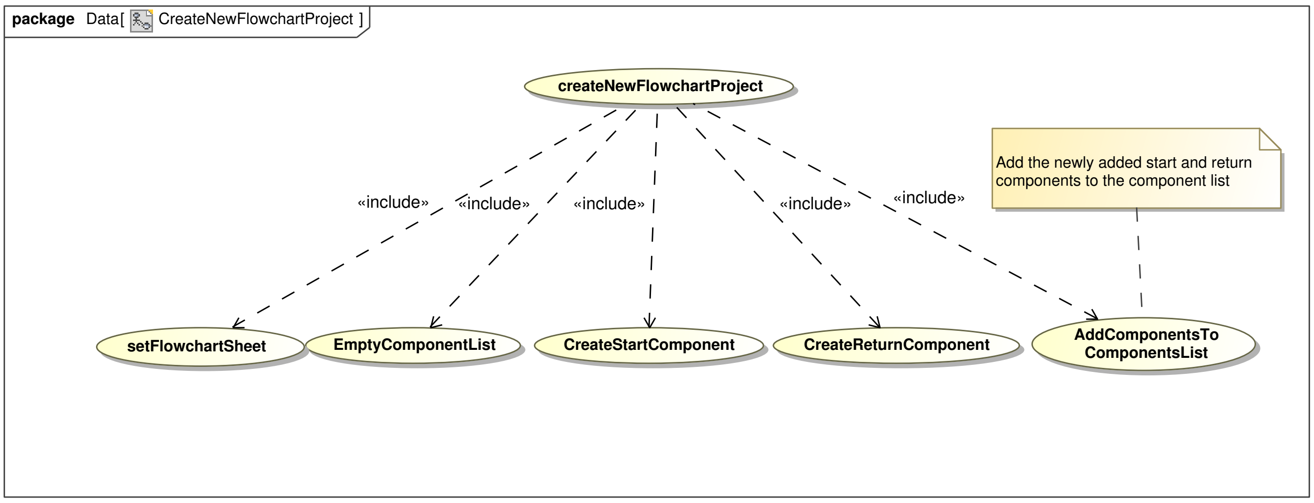

package Data[CreateNewFlowchartProject]
createNewFlowchartProject
«include»
«include»
«include»
«include»
«include»
Add the newly added start and return components to the component list
setFlowchartSheet
EmptyComponentList
CreateStartComponent
CreateReturnComponent
AddComponentsTo ComponentsList

Figure 1: createFlowchartProject Use Case Diagram

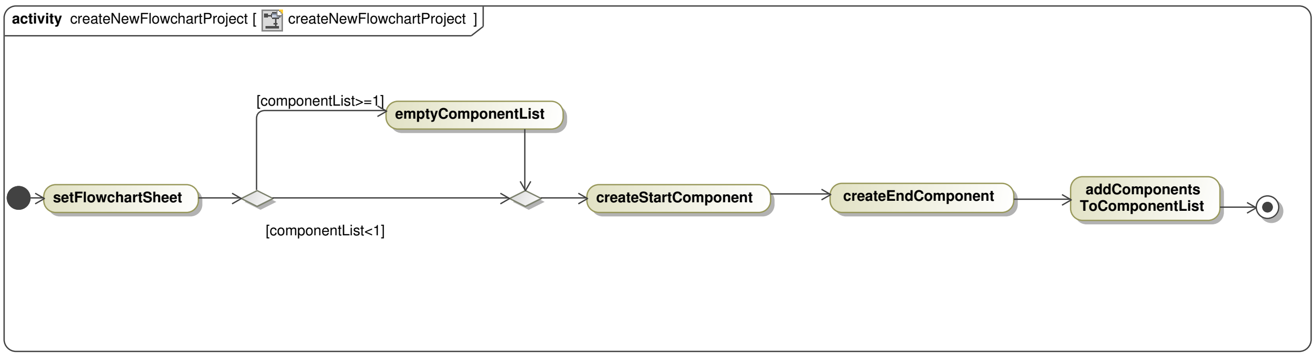


Figure 2: createFlowchartProject Activity Diagram

3.2 addFlowchartComponent

Provides users with functionality to select the flowchart components they want to add and place it on the planning canvas.

Pre Condition: Canvas is available and flowchart exist.

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

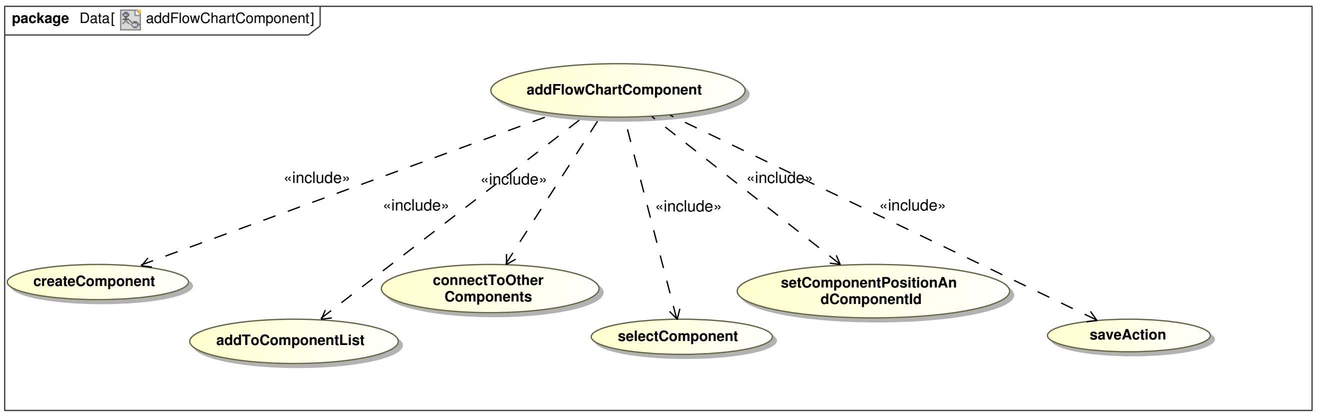


Figure 3: addFlowchartComponent Use Case Diagram

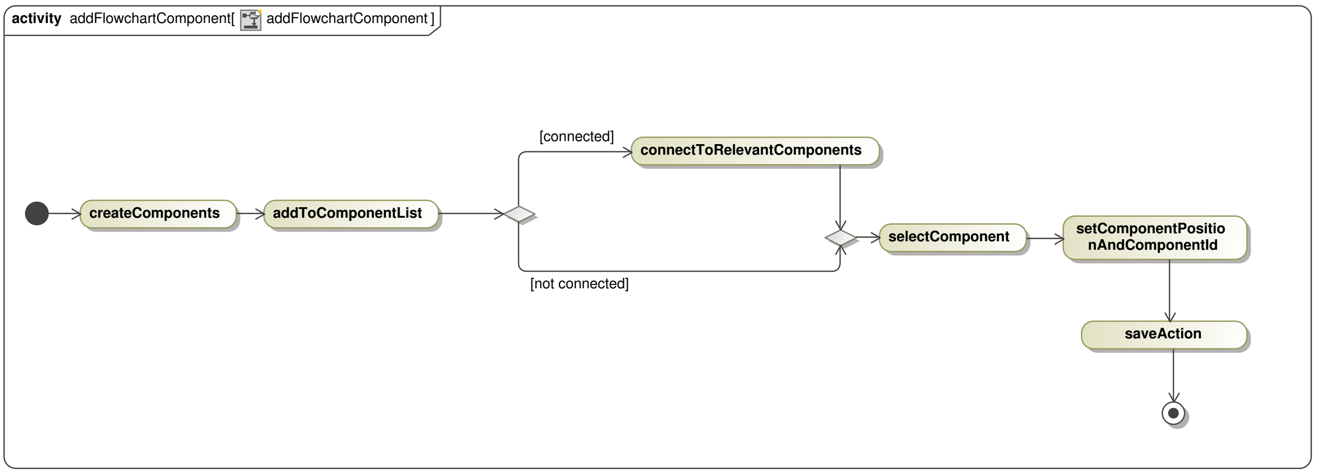

activity addFlowchartComponent[addFlowchartComponent]
createComponents
addToComponentList
[connected]
[not connected]
connectToRelevantComponents
selectComponent
setComponentPositionAndComponentId
saveAction

Figure 4: 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: Component must have been added to the canvas

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

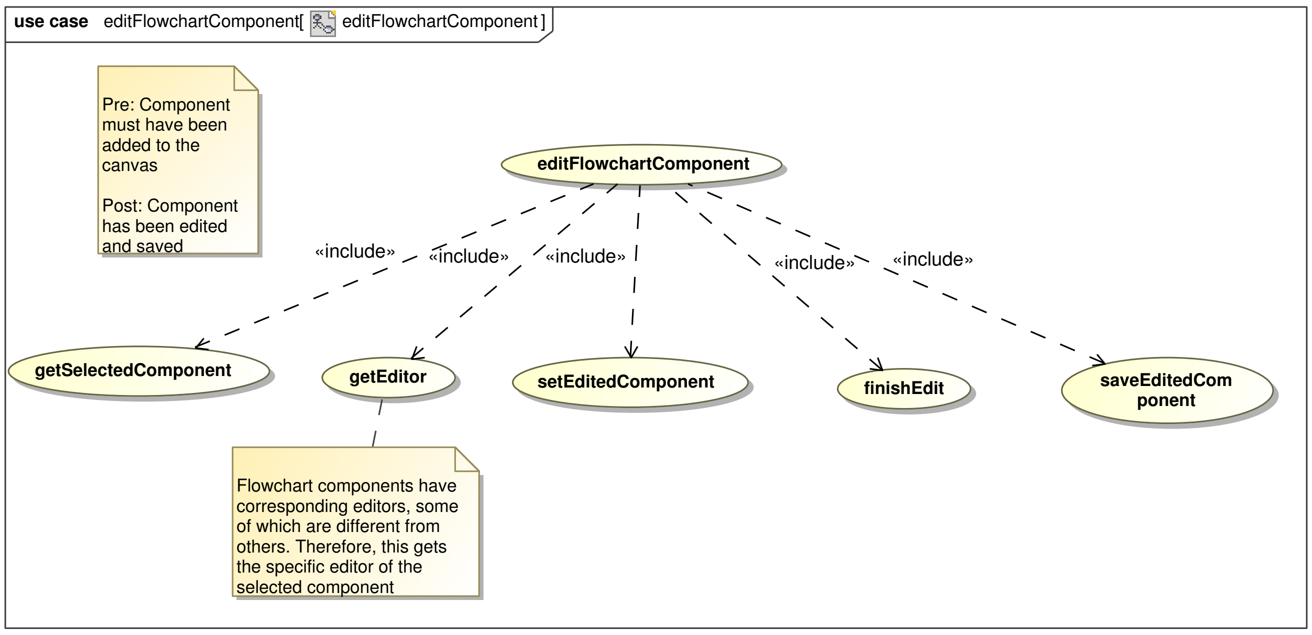

use case editFlowchartComponent[editFlowchartComponent]
Pre: Component must have been added to the canvas
Post: Component has been edited and saved
editFlowchartComponent
«include»
«include»
«include»
«include»
«include»
getSelectedComponent
getEditor
setEditedComponent
finishEdit
saveEditedComponent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

Figure 5: editFlowchartComponent Use Case Diagram

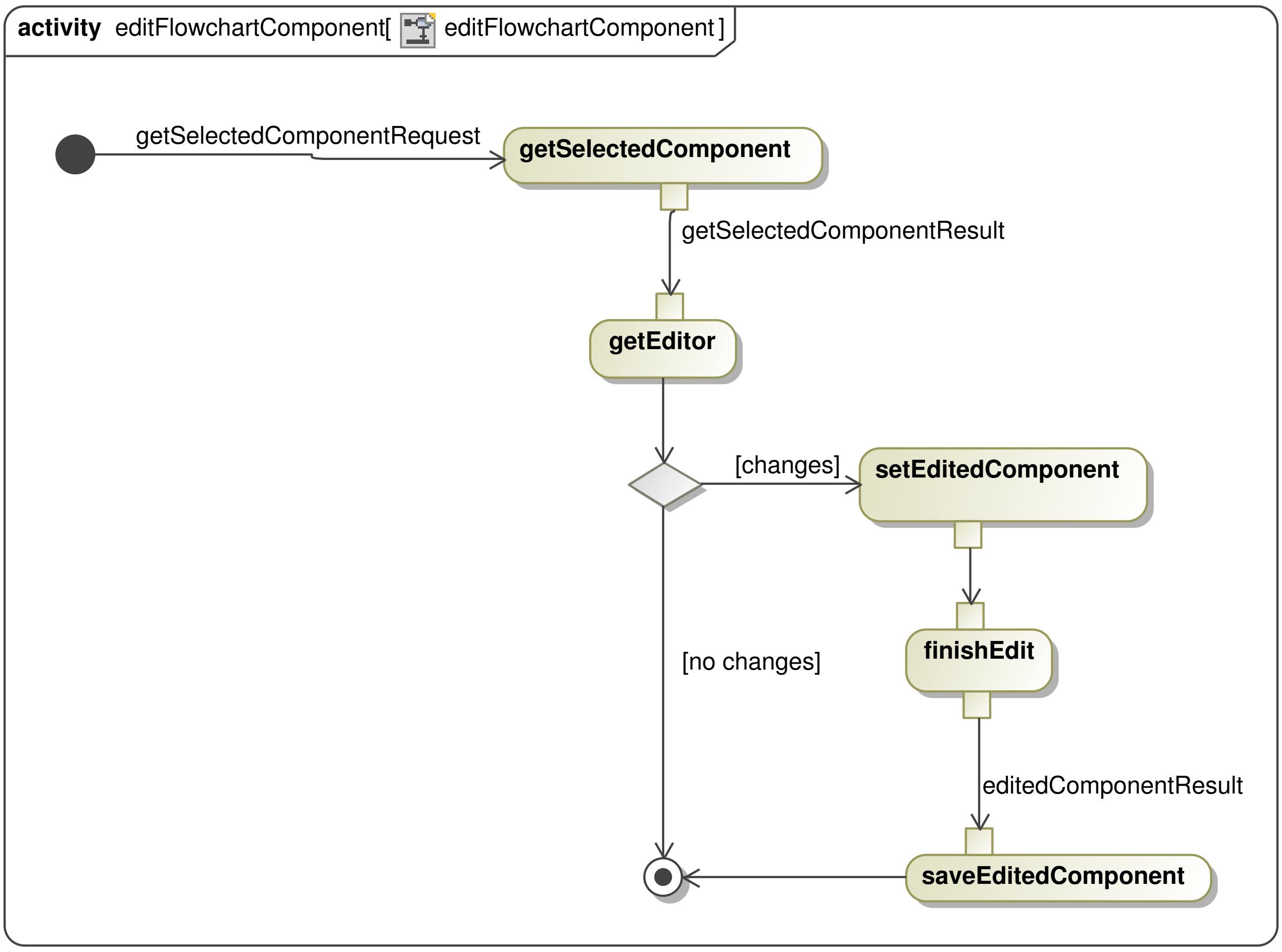


Figure 6: editFlowchartComponent Activity Diagram

3.4 deleteFlowchart

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

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 be active

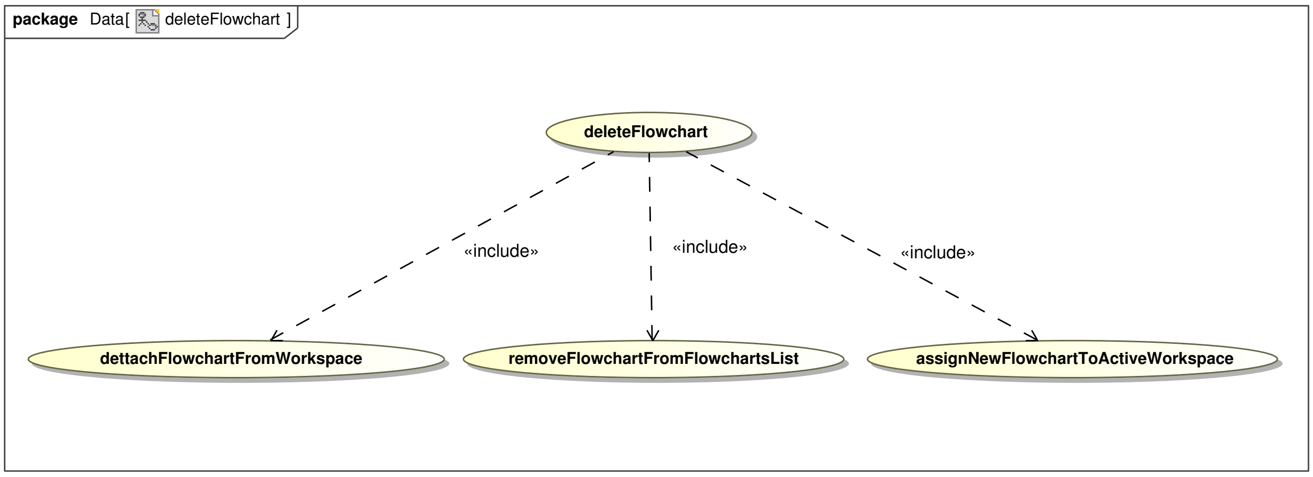

package Data[deleteFlowchart]
deleteFlowchart
«include»
«include»
«include»
dettachFlowchartFromWorkspace
removeFlowchartFromFlowchartsList
assignNewFlowchartToActiveWorkspace

Figure 7: deleteFlowchart Use Case Diagram

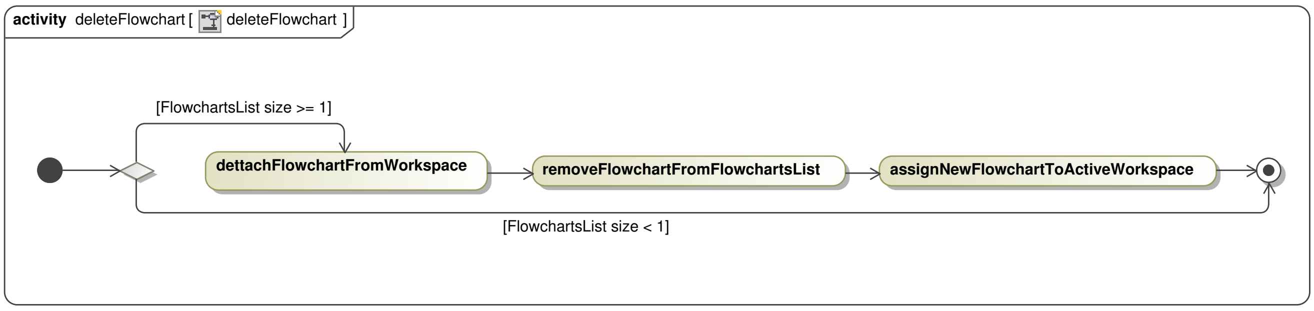

activity deleteFlowchart [deleteFlowchart]
[FlowchartsList size >= 1]
dettachFlowchartFromWorkspace
removeFlowchartFromFlowchartsList
assignNewFlowchartToActiveWorkspace
[FlowchartsList size < 1]

Figure 8: deleteFlowchart Activity Diagram

3.5 saveFlowchart

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

Pre Condition: Canvas is open

Post Condition: Flowchart has been saved to file

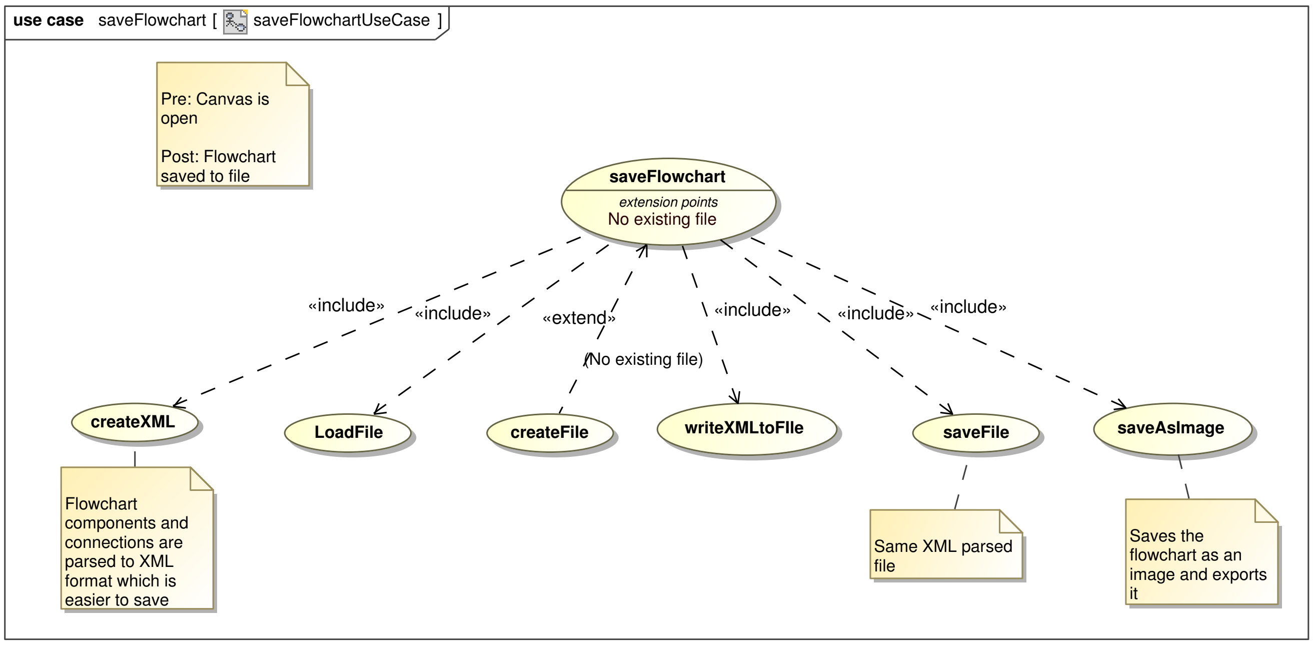


Figure 9: saveFlowchart Use Case Diagram

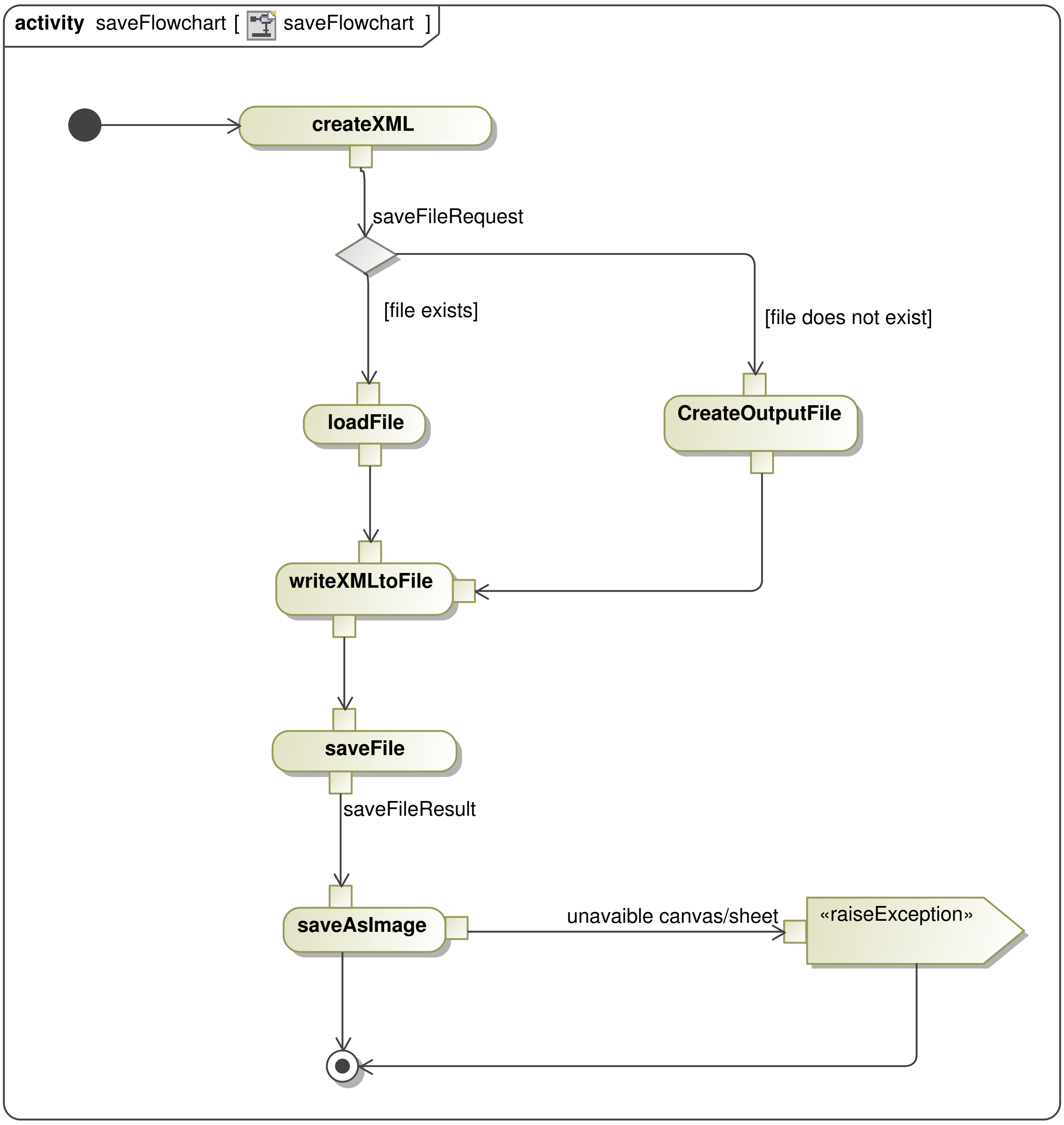


Figure 10: saveFlowchart Activity Diagram

3.6 loadFlowchart

loadFlowchart: users load flowcharts from existing files. The file is editable, can be modified by the user.

Pre Condition: File with correct extension already exists.

Post Condition: Open the file and make it editable in the canvas element.

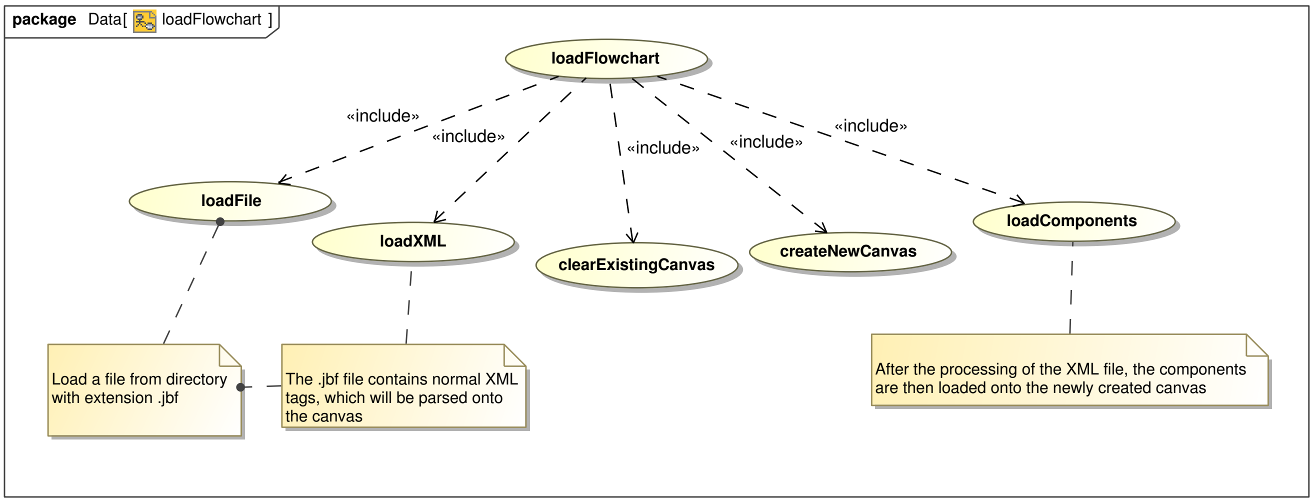

package Data[loadFlowchart]
loadFlowchart
«include»
«include»
«include»
«include»
«include»
loadFile
loadXML
clearExistingCanvas
createNewCanvas
loadComponents
Load a file from directory with extension .jbf
The .jbf file contains normal XML tags, which will be parsed onto the canvas
After the processing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

Figure 11: loadFlowchart Use Case Diagram

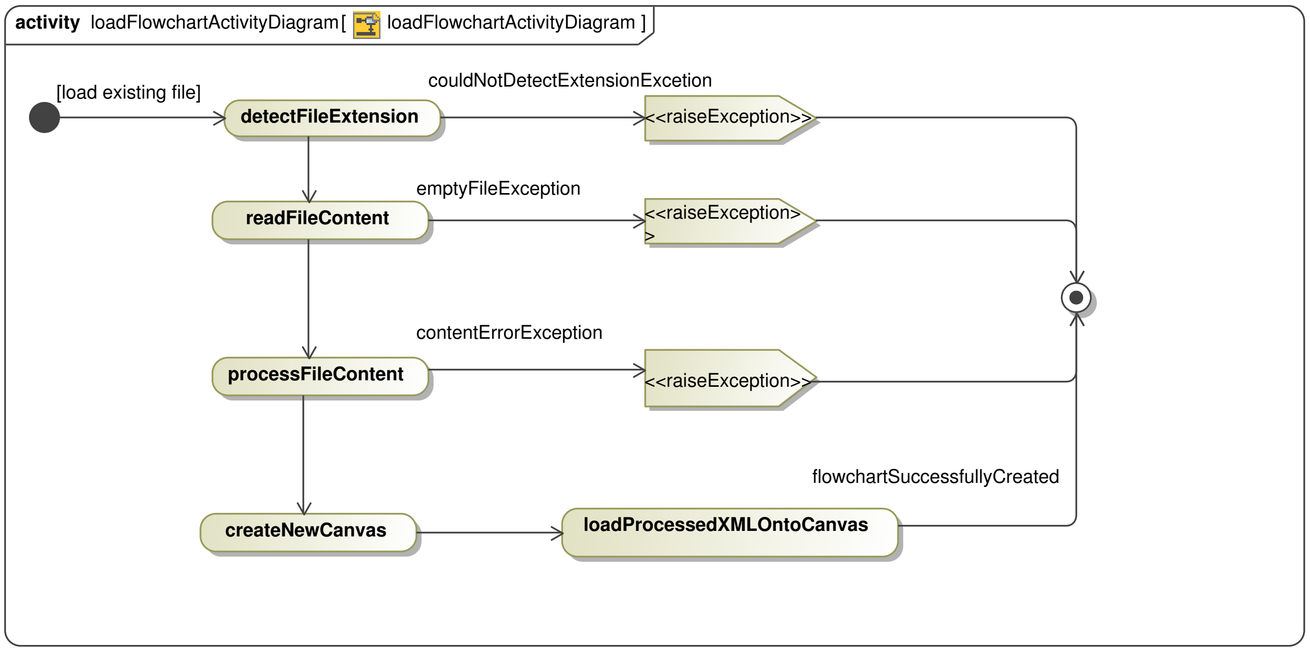


Figure 12: loadFlowchart Activity Diagram

3.7 deleteFlowchartComponent

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

Pre Condition: The canvas has to be available. 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.

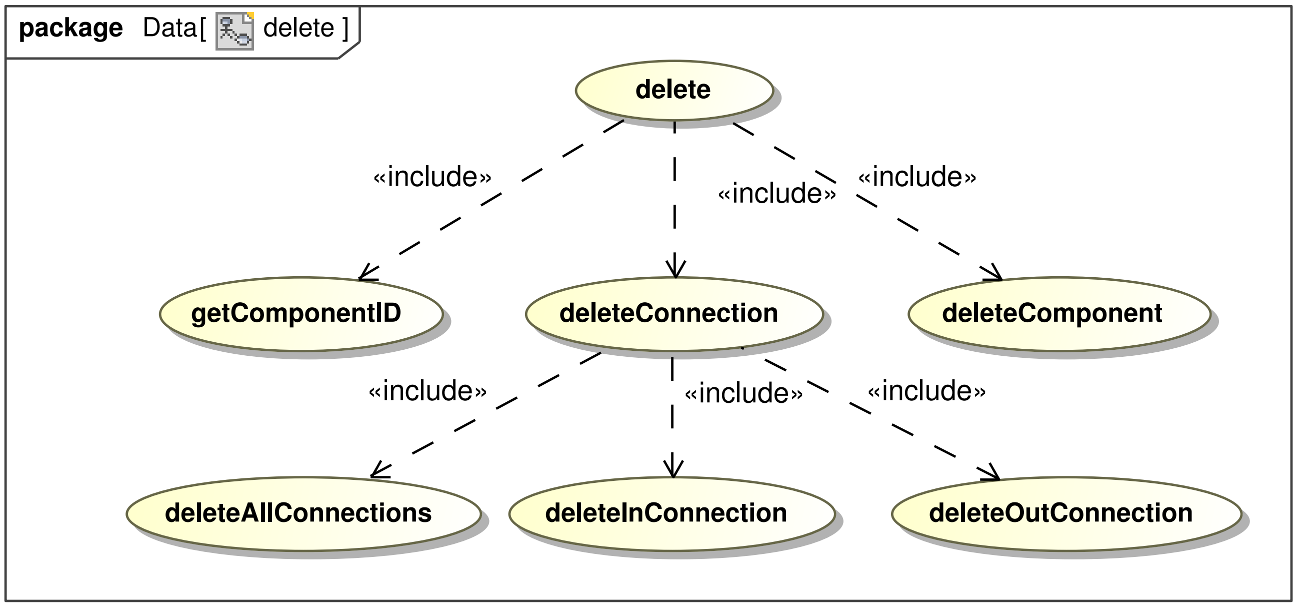


Figure 13: deleteFlowchartComponent Use Case Diagram

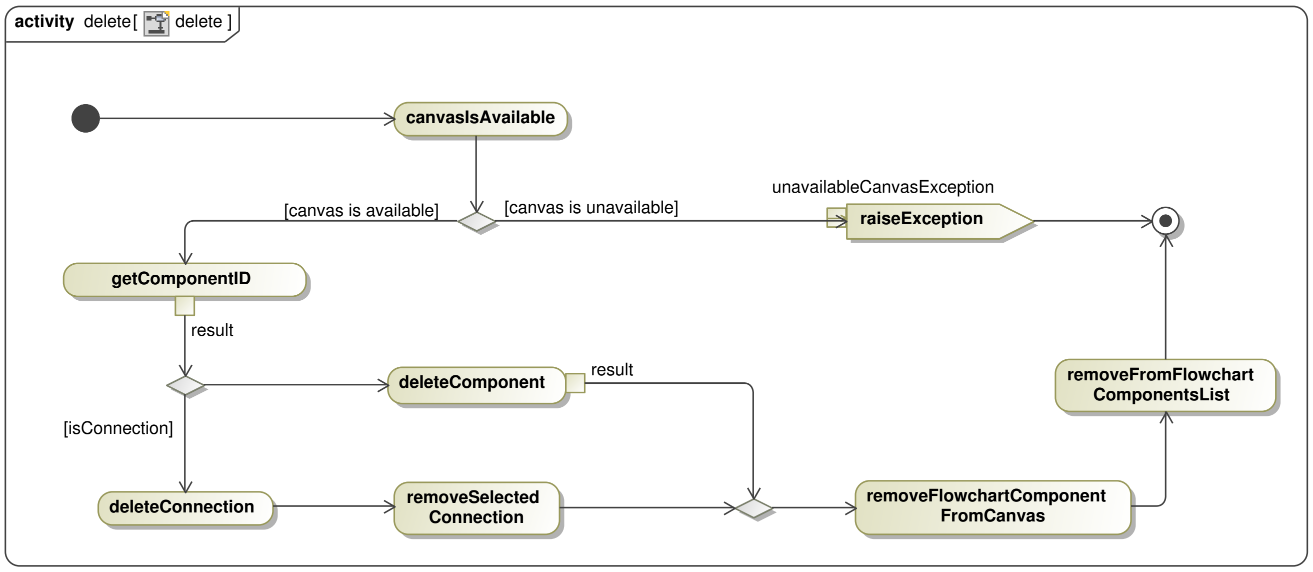


Figure 14: deleteFlowchartComponent Activity Diagram

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.

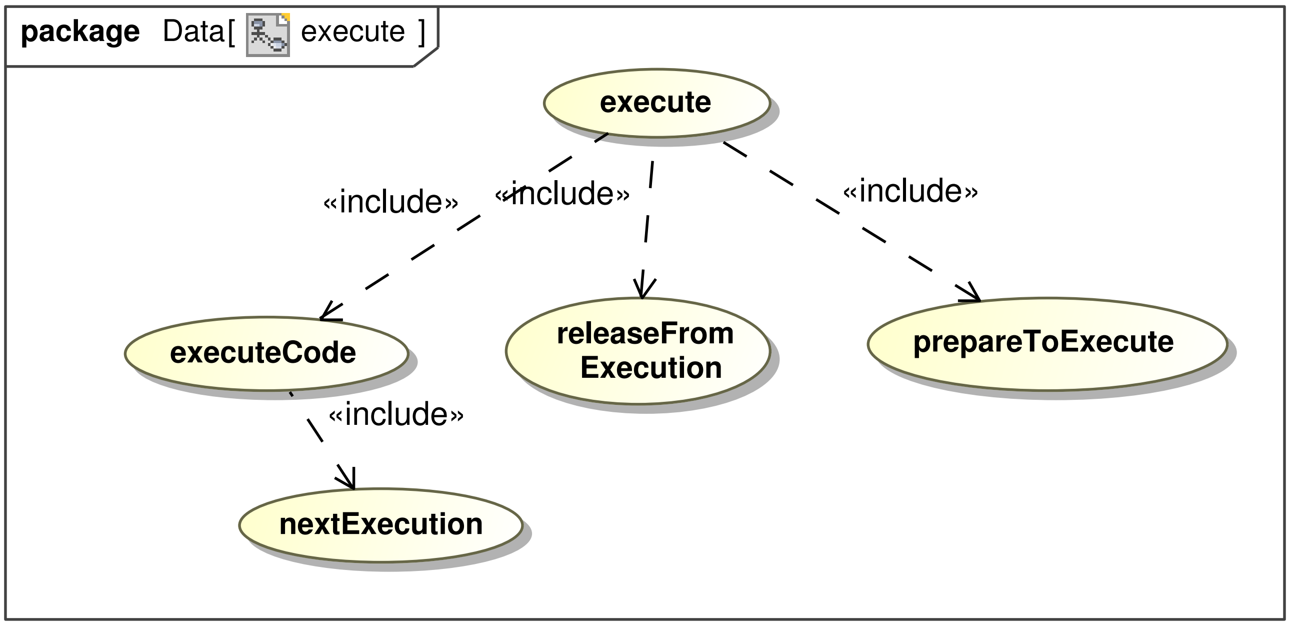


Figure 15: executeFlowchart Use Case Diagram

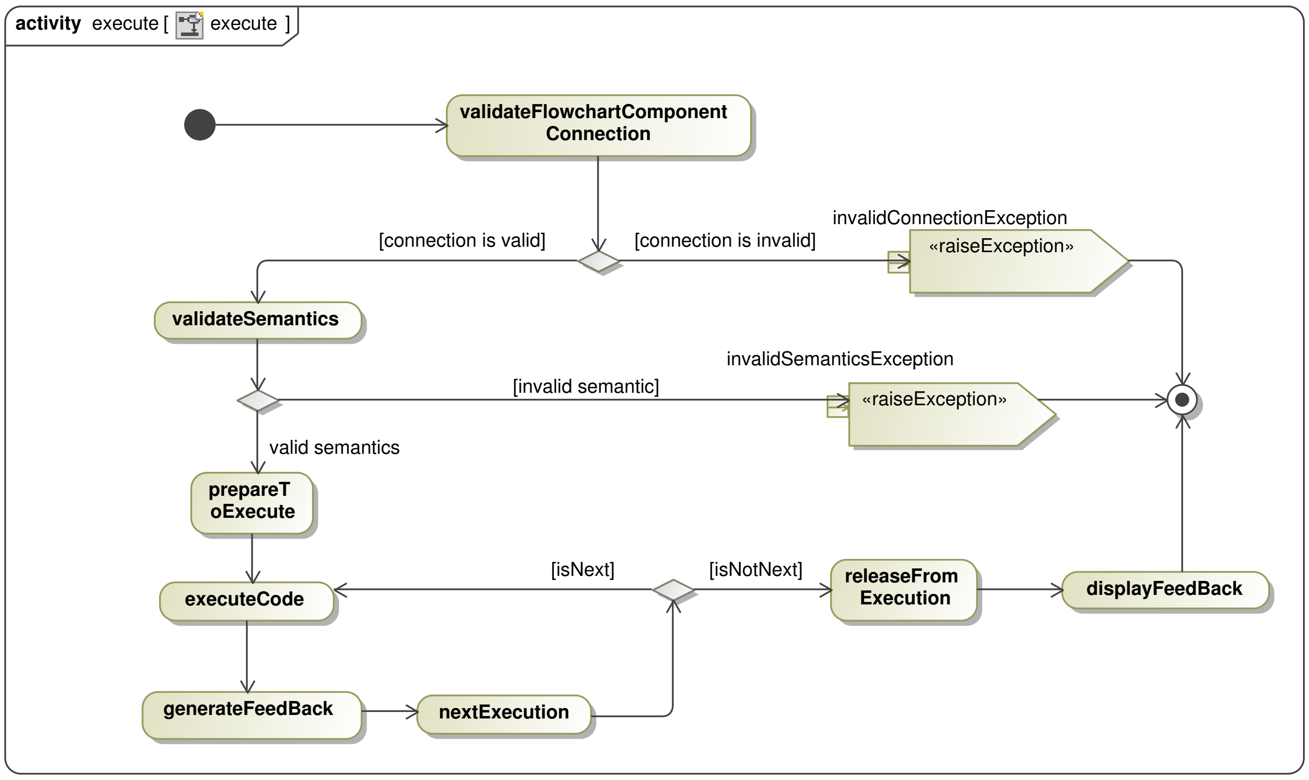


Figure 16: executeFlowchart Activity Diagram

4 The Domain Model - High-level

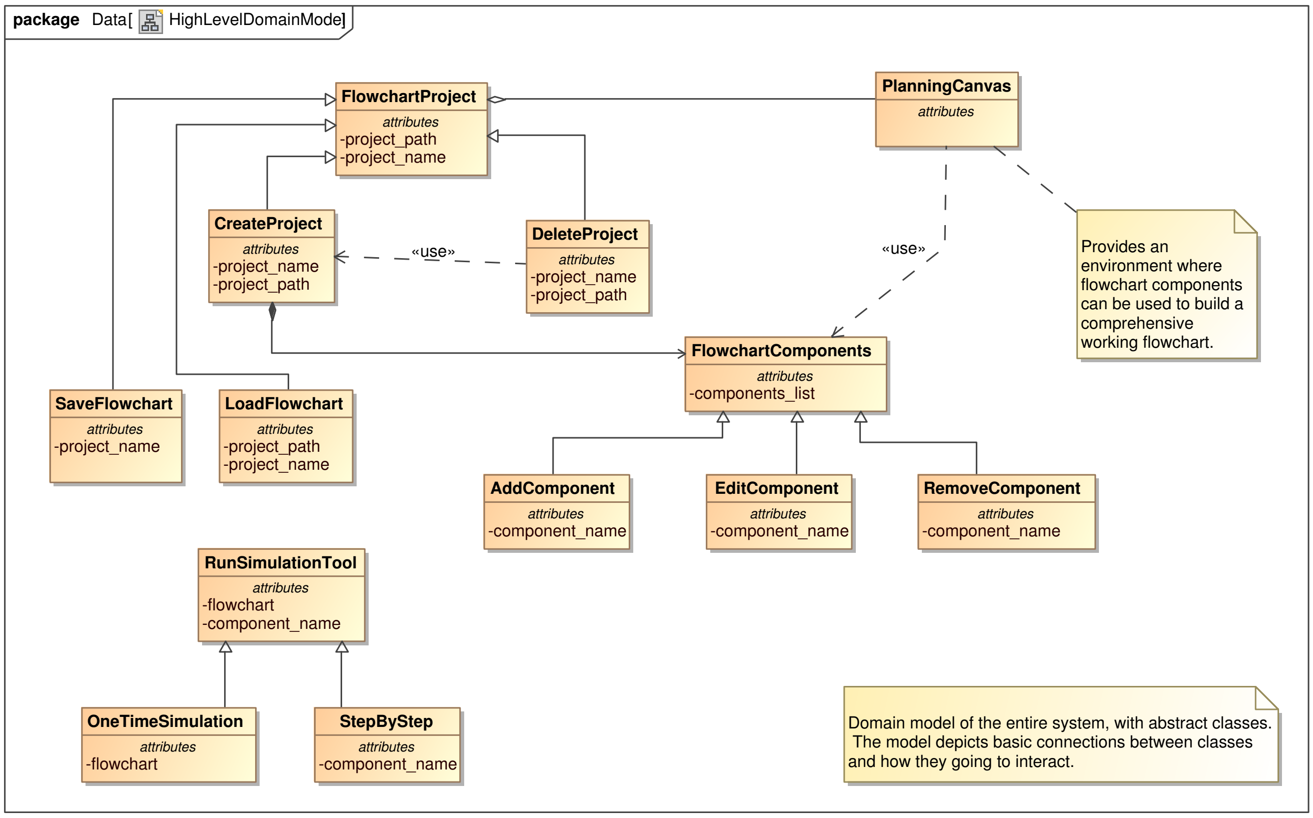


Figure 17: Domain Model Diagram

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 desktops running Linux operating system.

Security: Security is not a 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

Depending on which programming language best suits the interest of programmers, any language will be suitable. Preferably Java and C++. The system will resemble more or less the same programming styles used in Java and C++.

9 Technologies

For simplicity , Java would be recommended (Java is mostly used in data structures). Otherwise, any other programming language can be used.

Because the flowchart might need to be saved and used at the later stage, storage is another concern. XML will be used for storage. It can be easily parsed in Java and more convenient for serialization.