SYSTEM AND SOFTWARE DESIGN DESCRIPTION (SSDD): Incorporating Architectural Views and Detailed Design Criteria FOR

Phunctional UML Editor (pUML)

Version 1.0
May 10, 2012

Prepared for:
Dr. Clint Jeffery

Prepared by:
Josh Armstrong
Zach Curtis
Brian Bowles
Logan Evans
Jeremy Klas
Nathan Krussel
Maxine Major
Morgan Weir
David Wells
University of Idaho
Moscow, ID 83844-1010

CS383 SSDD
RECORD OF CHANGES (Change History)

Change Number	Date	Location of change (e.g., page or figure #)	A M D	Brief description of change	Initials
1	01/17/2012	SSDD	A	Added updated SSRS/SSDD pdf and TeX files	MM
2	02/01/2012	SSDD	A	Updated SSRS and SSDD	MM
3	02/13/2012	Section 4.1	M	Class diagram reflects node factory addition	MM
4	05/08/2012	SSDD	M	Document overhaul including sections, references, and class diagrams	MM
5	05/10/2012	SSDD	M	Interaction diagrams section overhaul	MM

A - ADDED M - MODIFIED D – DELETED

Phunctional UML Editor
TABLE OF CONTENTS

Section Page

1 INTRODUCTION

1.1 IDENTIFICATION

This document is a stand-alone document and has no identification numbers other than the revision number.

1.2 DOCUMENT PURPOSE, SCOPE, AND INTENDED AUDIENCE

1.2.1 Document Purpose

Phunctional UML Editor software is being developed according to a set of requirements outlined in the pUML System and Software Requirements Specification (Rev. 1.0). This document will provide detailed information regarding the design implementation of these requirements.

1.2.2 Document Scope and/or Context

This document includes information regarding the design and components of the pUML software. The class structure and interactions to implement features, along with rationale for these design decisions is provided.

1.2.3 Intended Audience for Document

This document may be referenced for educational purposes by Computer Science students and faculty at the University of Idaho.

1.3 SYSTEM AND SOFTWARE PURPOSE, SCOPE, AND INTENDED USERS

1.3.1 System and Software Purpose

The pUML software is intended to be a tool utilized by software designers to create UML diagrams.

1.3.2 System and Software Scope/or Context

The pUML software will be designed to provide functionality to create UML diagram projects. Users will be able to create several different types of UML diagrams, create, modify, link, save, and delete objects within individual UML diagrams, and save collections of diagrams stored as part of a project.

1.3.3 Intended Users for the System and Software

The completed product would be available to the general public for purchase, however this specific release will be intended strictly for use by the University of Idaho Computer Science Department students and faculty, for educational purposes only.

1.4 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

Term or Acronym	Definition
AD	Architectural Description: "A collection of products to document an architecture" ISO/IEC 42010:2007 (§3.4).
Alpha test	Limited release(s) to selected, outside testers
Architectural Description	(AD) "A collection of products to document an architecture" ISO/IEC 42010:2007 (§3.4).
Architectural View	"A representation of a whole system from the perspective of a related set of concerns" ISO/IEC 42010:2007 (§3.9).
Architecture	"The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" ISO/IEC 42010:2007 (§3.5).
Beta test	Limited release(s) to cooperating customers wanting early access to developing systems
Design Entity	"An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced" IEEE STD 1016-1998 (§3.1).
Design View	"A subset of design entity attribute information that is specifically suited to the needs of a software project activity" IEEE STD 1016-1998 (§3.2).
Final test	aka, Acceptance test, release of full functionality to customer for approval
DFD	Data Flow Diagram
SSDD	System and Software Design Document
SSRS	System and Software Requirements Specification
System	"A collection of components organized to accomplish a specific function or set of functions" ISO/IEC 42010:2007 (§3.7).
System and Software Architecture and Design Description	An architectural and detailed design description that includes a software system within the context of its enclosing system and describes the enclosing system, the enclosed software, and their relationship and interfaces.

1.5 DOCUMENT REFERENCES

1) CSDS, *System and Software Requirements Specification Template*, Version 1.0, July 31, 2008, Center for Secure and Dependable Systems, University of Idaho, Moscow, ID, 83844.

2) ISO/IEC/IEEE, *IEEE Std 1471-2000 Systems and software engineering – Recommended practice for architectural description of software intensive systems,* First edition 2007-07-15, International Organization for Standardization and International Electrotechnical Commission, (ISO/IEC), Case postale 56, CH-1211 Genève 20, Switzerland, and The Institute of Electrical and Electronics Engineers, Inc., (IEEE), 445 Hoes Lane, Piscataway, NJ 08854, USA.

3) IEEE, *IEEE Std 1016-1998 Recommended Practice for Software Design Descriptions*, 1998-09-23, The Institute of Electrical and Electronics Engineers, Inc., (IEEE) 445 Hoes Lane, Piscataway, NJ 08854, USA.

4) 3) ISO/IEC/IEEE, *IEEE Std. 15288-2008 Systems and Software Engineering – System life cycle processes,* Second edition 2008-02-01, International Organization for Standardization and International

Electrotechnical Commission, (ISO/IEC), Case postale 56, CH-1211 Genève 20, Switzerland, and The Institute of Electrical and Electronics Engineers, Inc., (IEEE), 445 Hoes Lane, Piscataway, NJ 08854, USA.

5) ISO/IEC/IEEE, IEEE Std. 12207-2008, *Systems and software engineering – Software life cycle processes,* Second edition 2008-02-01, International Organization for Standardization and International Electrotechnical Commission, (ISO/IEC), Case postale 56, CH-1211 Genève 20, Switzerland, and The Institute of Electrical and Electronics Engineers, Inc., (IEEE), 445 Hoes Lane, Piscataway, NJ 08854, USA.

1.6 DOCUMENT OVERVIEW

Section 2 of this document describes the system and software constraints imposed by the operational environment, system requirements and user characteristics, and then identifies the system stakeholders and lists describes their concerns and mitigations to those concerns.

Section 3 of this document describes the system and software architecture from several viewpoints, including, but not limited to, the developer's view and the user's view.

Section 4 provides detailed design descriptions for every component defined in the architectural view(s). Sections 5 provides traceability information connecting the original specifications (referenced above) to the architectural components and design entities identified in this document.

Section 6 and beyond are appendices including original information and communications used to create this document.

1.7 DOCUMENT RESTRICTIONS

This document is for LIMITED RELEASE ONLY to UI CS personnel working on the project.

2 CONSTRAINTS AND STAKEHOLDER CONCERNS

2.1 CONSTRAINTS

2.1.1 Environmental Constraints.

The pUML software poses no environmental constraints at this time .

2.1.2 System Requirement Constraints.

The pUML software will be designed to function on, Windows 7, and Linux. Cross platform functionality will minimize portability errors and allow for projects to be migrated between platforms with minimal difficulty. However, the pUML software is not intended to be migrated to any other platforms with any guaranteeable level of functionality.
The pUML software is also not intended to be utilized by multiple users.
This release will not include several features which may be industry standard for UML diagram editors. These features would be incorporated into a later software release.

2.1.3 User Characteristic Constraints.

University of Idaho Computer Science students and faculty should be able to reasonably understand and operate the pUML software.

2.2 STAKEHOLDER CONCERNS

There are no stakeholders for our software at this time.

3 SYSTEM AND SOFTWARE ARCHITECTURE

3.1 DEVELOPER'S ARCHITECTURAL VIEW

3.1.1 Developer's View Identification

This is the architecture of the program from the viewpoint of the developer. The purpose is to give an overview of the details of the major components of the architecture.

In order to have the program be able to draw diagrams, a custom QWidget is defined called the Canvas. The Canvas holds all the instantiations of nodes and draws each one. It also creates the nodes by handling the mouse click events. The toolbar and menu system lets the Canvas know which type of object will be created next. The Canvas is a member of the MainWindow class, which inherits from QMainWindow.

3.1.2 Developer's View Representation and Description

The Canvas contains a vector container of ObjectNodes. Each ObjectNode has a draw function which takes a QPainter reference as an argument and draws the appropriate figure with the QPainter. The program then defines it's own ObjectNodes, e.g. CircleNode and DiamondNode, and pushes them into the vector. In this way the Canvas can draw each of the nodes in the diagram. To create a new object, it handles a mouse click event and creates a new object of the type specified by a previous call to it's function to set a new object type. The new object is pushed into the vector and then the draw function is called on every node in that vector. When selecting a node to edit or delete, the Canvas takes the X and Y coordinates of the click and translates that into the index of the object selected. Then the Canvas can popup a menu to edit the node or delete the node.

3.1.3 Developer's Architectural Rationale

We decided to create a new QWidget for the Canvas so that it can handle click events and have a paint function. We then decided to have the nodes represented by a vector so that it can be easily iterated over and quickly accessed by index. We decided to have the nodes be represented by specific definitions of ObjectNodes so that they can all be pushed into the a vector ObjectNodes. This way each of the nodes can define their own draw function, as well private data such as radius for circles. This allows new objects to be easily created.

3.2 USER'S ARCHITECTURAL VIEW

3.2.1 User's View Identification

This is the viewpoint of the program from the viewpoint of the user. From this viewpoint, there are three major components of the program: the Canvas, the Toolbar and the Menu.

3.2.2 User's View Representation and Description

The menu and the toolbar have redundant functionality. The toolbar provides quick access to certain menu items, such as available objects and connectors. The canvas provides a space for the user to place objects and connectors during creation of a UML diagram. The pUML software also provides options for the user to save and load pUML UML diagrams.

3.3 CONSISTENCY OF ARCHITECTURAL VIEWS

Due to the limited scope of this project, there are no known inconsistencies between views.

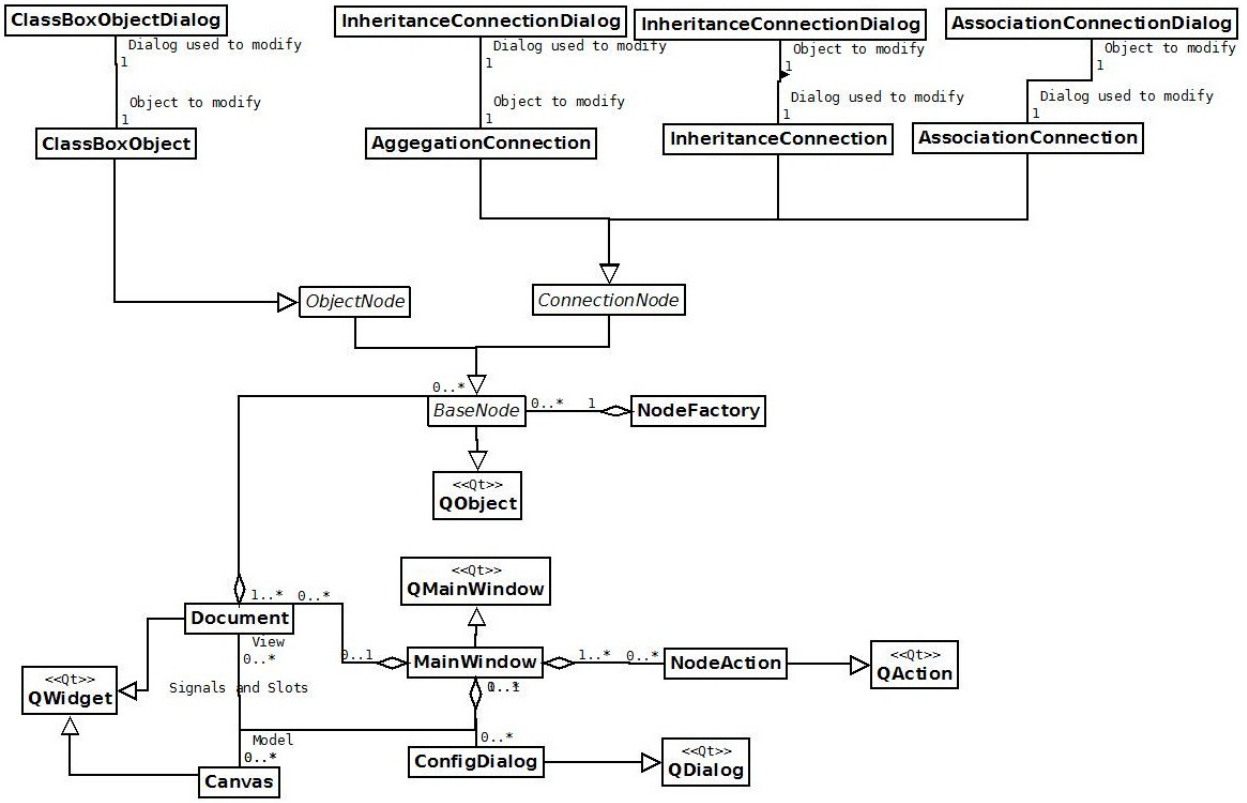
4 SOFTWARE DETAILED DESIGN

4.1 DEVELOPER'S VIEWPOINT DETAILED SOFTWARE DESIGN

Diagrams depicting classes and relationships between classes for the pUML software are provided in this section.

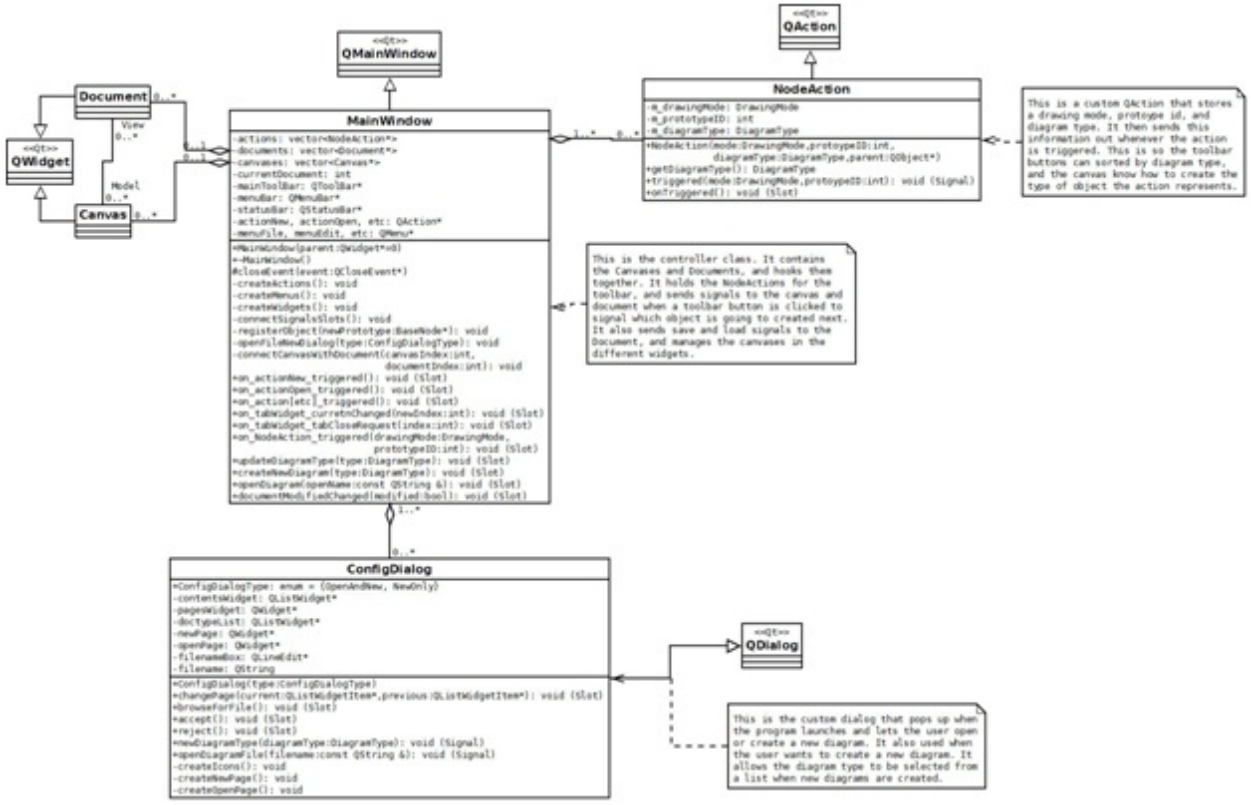
4.1.1 Class Overview

This is a simplified version of the full class diagram for the pUML software. This diagram shows the names of the main classes and the data flow between them.
The nodes dependent on BaseNode are examples of node types, in this case taken from the UML Class Diagram type. Each diagram type consists of objects and connectors, each of which are designed according to this template.



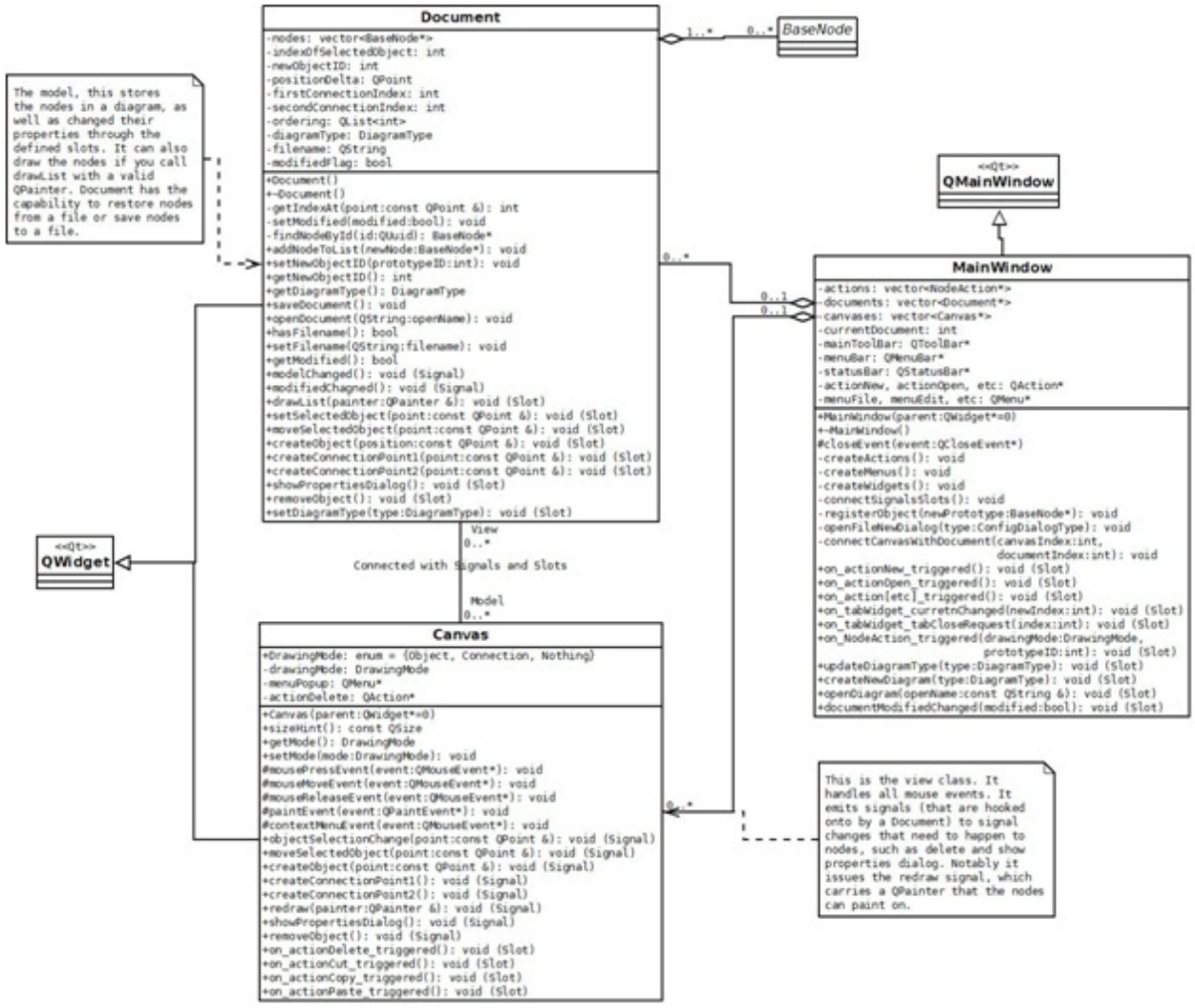
4.1.2 Main Window Class

The MainWindow class is the controller class, joining the Document and Canvas together. This class brings together all the components of pUML at the largest scale. This class is also responsible for guiding the process by which the various nodes appear on the canvas.



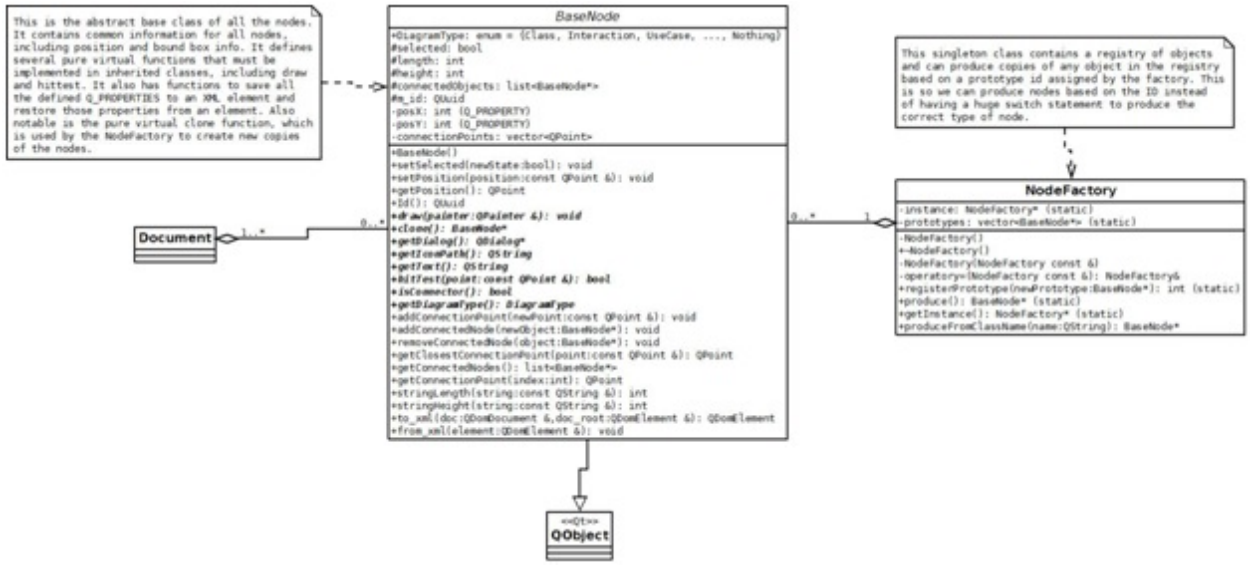
4.1.3 Document and Canvas Classes

In the Model-View system, the Document performs as the model, controlling all information about the individual nodes in a diagram, and is critical in the process of saving and loading.
The Canvas class is responsible for the view, and handles all mouse events and actions related to the mouse events, including the properties dialog and node deletion.



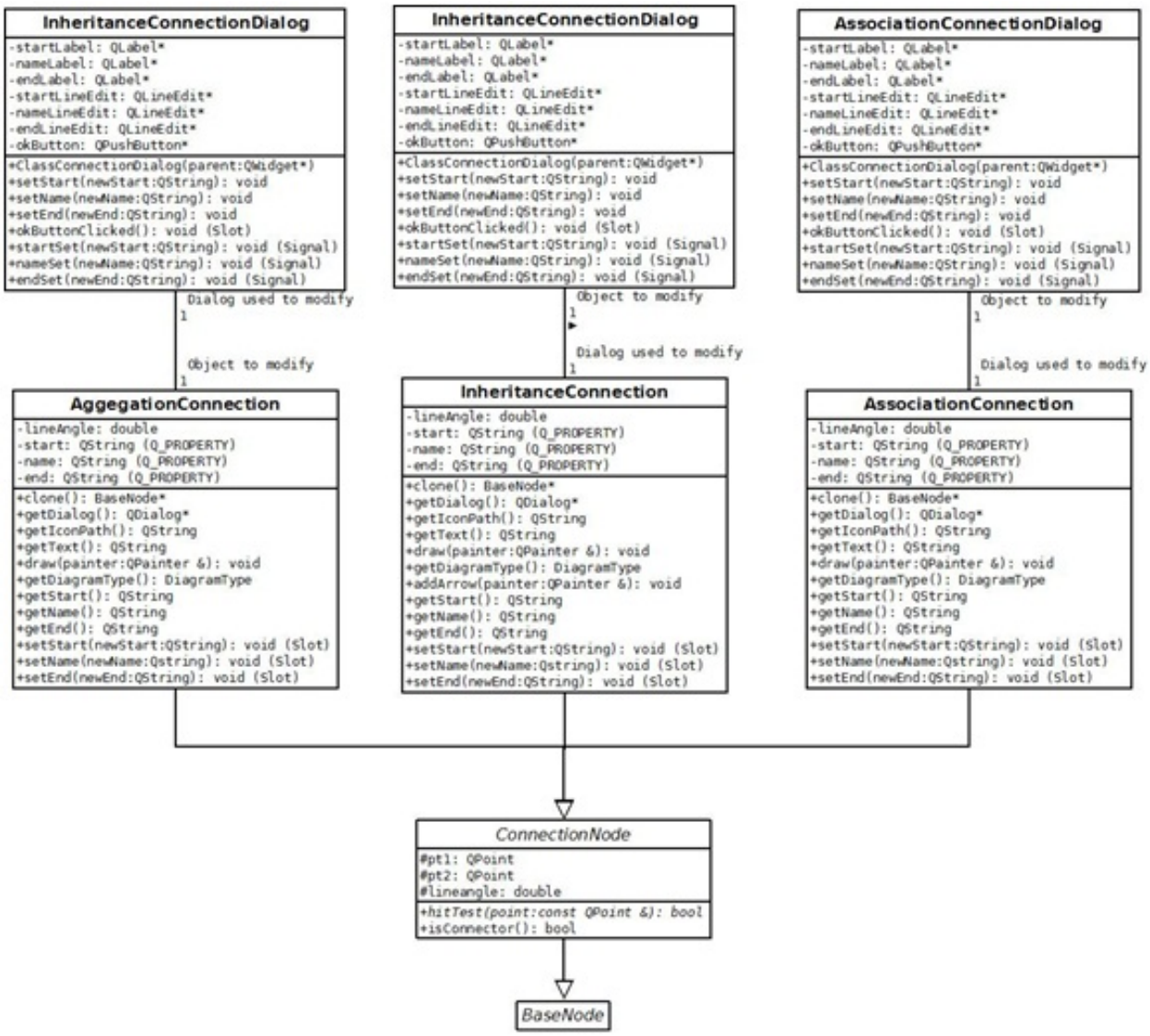
4.1.4 Base Node Class

The BaseNode is the most abstract node class, containing all the standard information for all nodes. This class also incorporates the cloning function, which is critical in developing several copies of any type of node. The NodeFactory contains a registry of objects, and may produce copies of these objects. Together with BaseNode, each node placed on the canvas in a UML diagram is guaranteed to be purely unique, even as an exact replica of another node.



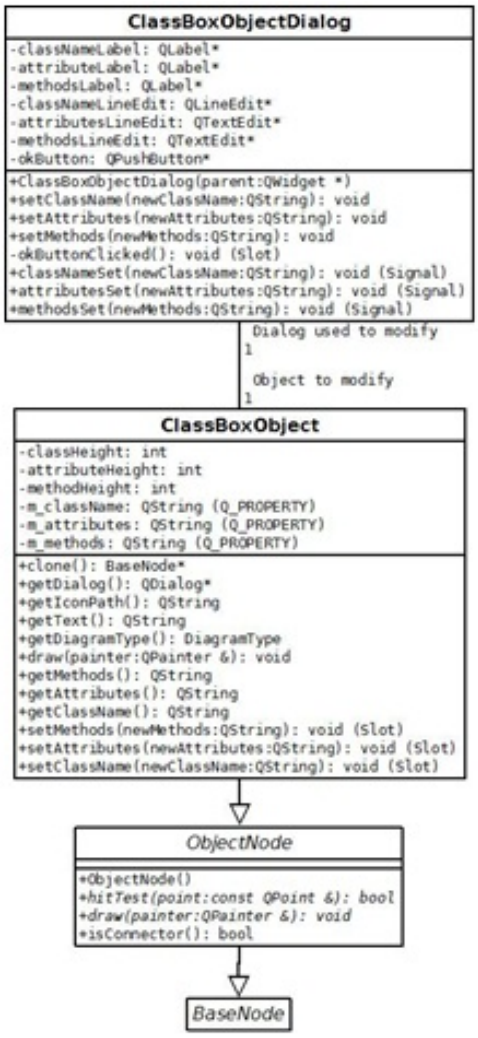
4.1.5 ConnnectionNode Class

The ConnectionNode class incorporates both specific types of connection and the dialogs unique to each connection type. This allows for several types of connections, unique with their own dialog properties.
The example shown is from a Class diagram. Connections from other diagram types would have a nearly identical format, with minor diagram-specific differences.



4.1.6 Object Node Class

The ObjectNode class incorporates specific object types and dialogs unique to each object type. This allows for several types of objects, unique with their own dialog properties.
The example shown is from a Class diagram. Objects from other diagram types would have a nearly identical format, with minor diagram-specific differences.



4.2 COMPONENT/ENTITY DICTIONARY

This table shows the individual components, implemented as classes, utilized in the development of this software, and how each class is related to the other classes.

Component/Entity Dictionary				
Name of Class	**Type/Range**	**Purpose/Function**	**Dependencies**	**Subordinates**
QMainWindow	QWidget QMainWindow	Connects the canvas and document together	QMainWindow	ConfigDialog NodeAction
ConfigDialog	QDialog	Main dialog which allows the user to select a new diagram type or open an existing diagram.	MainWindow	N/A
NodeAction	QAction	Sets the drawing mode	Main Window	N/A
Canvas	QWidget	Draws objects and connectors	Main Window	N/A
Document	QWidget	Contains all diagram information	MainWindow	N/A
BaseNode	QObject	Draws the object	Document	ObjectNode ConnectionNode
NodeFactory	N/A	Registry of objects, which may be copied by ID	BaseNode	N/A
ObjectNode	BaseNode	Contains object	BaseNode	ObjectNode ConnectionNode
*ObjectType	BaseNode	Contains object features	ObjectNode	*ObjectTypeDialog
*ObjectTypeDialog	QDialog	Contains object dialog	*ObjectType	N/A
ConnectionNode	BaseNode	Contains connection	BaseNode	*ConnectionType
*ConnectionType	BaseNode	Contains connection type	ConnectionNode	*ConnectionTypeDialog
*ConnectionTypeDialog	QDialog	Contains connection features	*ConnectionType	N/A

* Object and Connection types are varied, but follow this pattern.
The names of the respective classes vary per diagram type and object/connector type, but follow the same template. ** All QT classes are not detailed in this document.

4.3 FEATURE DETAILED DESIGN

Each of the diagrams in this section represent an important functionality of the pUML software. The interaction diagrams detail the object classes that will be utilized in the execution of each of these major features of the software, as well as show how these classes interact.

4.3.1 Detailed Design for Feature: Create New Diagram

4.3.1.1 Introduction/Purpose of this Feature Each of the UML diagrams will need its own space for development within the pUML software. The "New" feature allows the user to select a diagram and start developing that diagram on a blank, diagram-specific canvas.

4.3.1.2 Input for this Feature The user selects "New" from the main menu options, and selects a diagram type from the Configuration Dialog box.

4.3.1.3 Output for this Feature pUML will then load a blank canvas in a new tab, with a toolbar containing only that diagram's type of objects and connectors.

4.3.1.4 Feature Process to Convert Input to Output User selects "New" from main menu. MainWindow class creates a ConfigDialog box, which contains all possible diagram types. User selects diagram type. ConfigDialog returns the diagram type back to the MainWindow class, which in turn uses that information to create both the Canvas and Document classes. MainWindow then connects the Canvas and Document classes, and sends the user-selected diagram type to the Document class. The main window then creates a new tab with a blank canvas and all diagram-specific properties.

4.3.1.5 Design Constraints and Performance Requirements of this Feature New diagrams may not be of a generic diagram type. The user must specify what type of a new diagram they will be developing prior to pUML creating the space in which to develop a UML diagram.

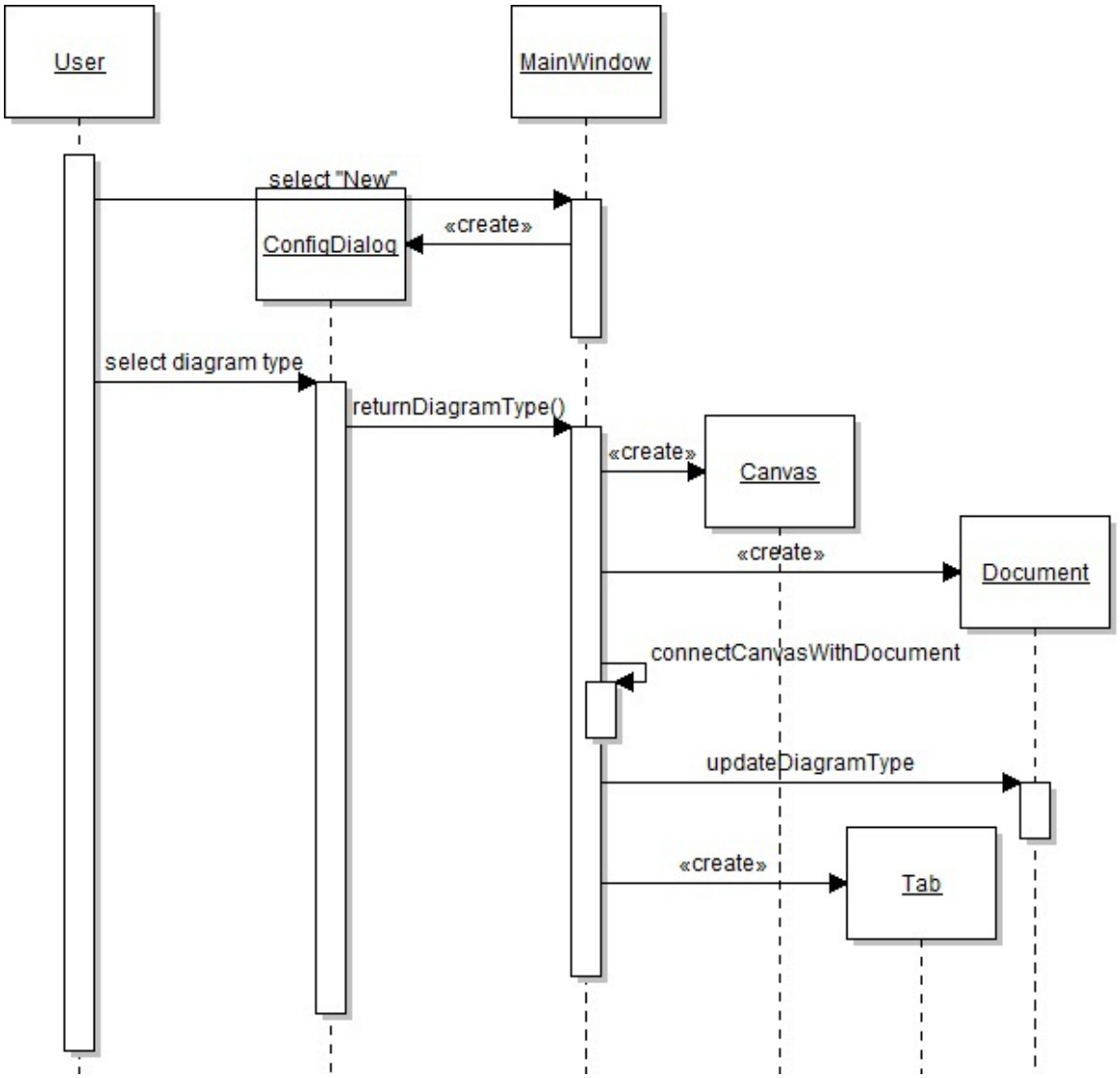


Figure 1: Create New Diagram

4.3.2 Detailed Design for Feature: Open an Existing Diagram

4.3.2.1 Introduction/Purpose of this Feature The pUML software provides the Open feature so that users may store diagrams and view or edit them later. Open will load the .puml diagram type into pUML, which will allow the user to continue modification.

4.3.2.2 Input for this Feature User selects "Open" from the main menu. The user is then prompted to locate the file on their computer.

4.3.2.3 Output for this Feature If the user-selected file is of .puml type, pUML will load the diagram into a new tab of that particular diagram type, and the diagram will be modifiable.

4.3.2.4 Feature Process to Convert Input to Output User selects "Open" from the MainWindow class, which creates the OpenFileDialog. The User then selects which file they choose to open, and OpenFileDialog returns that value to MainWindow. MainWindow creates both the Canvas and Document classes and then connects these two classes. MainWindow creates the tab for the diagram to load into, and sends diagram specific information to the Document class. The Document class then returns the diagram type to MainWindow.

4.3.2.5 Design Constraints and Performance Requirements of this Feature The pUML software will only open files of .puml type.

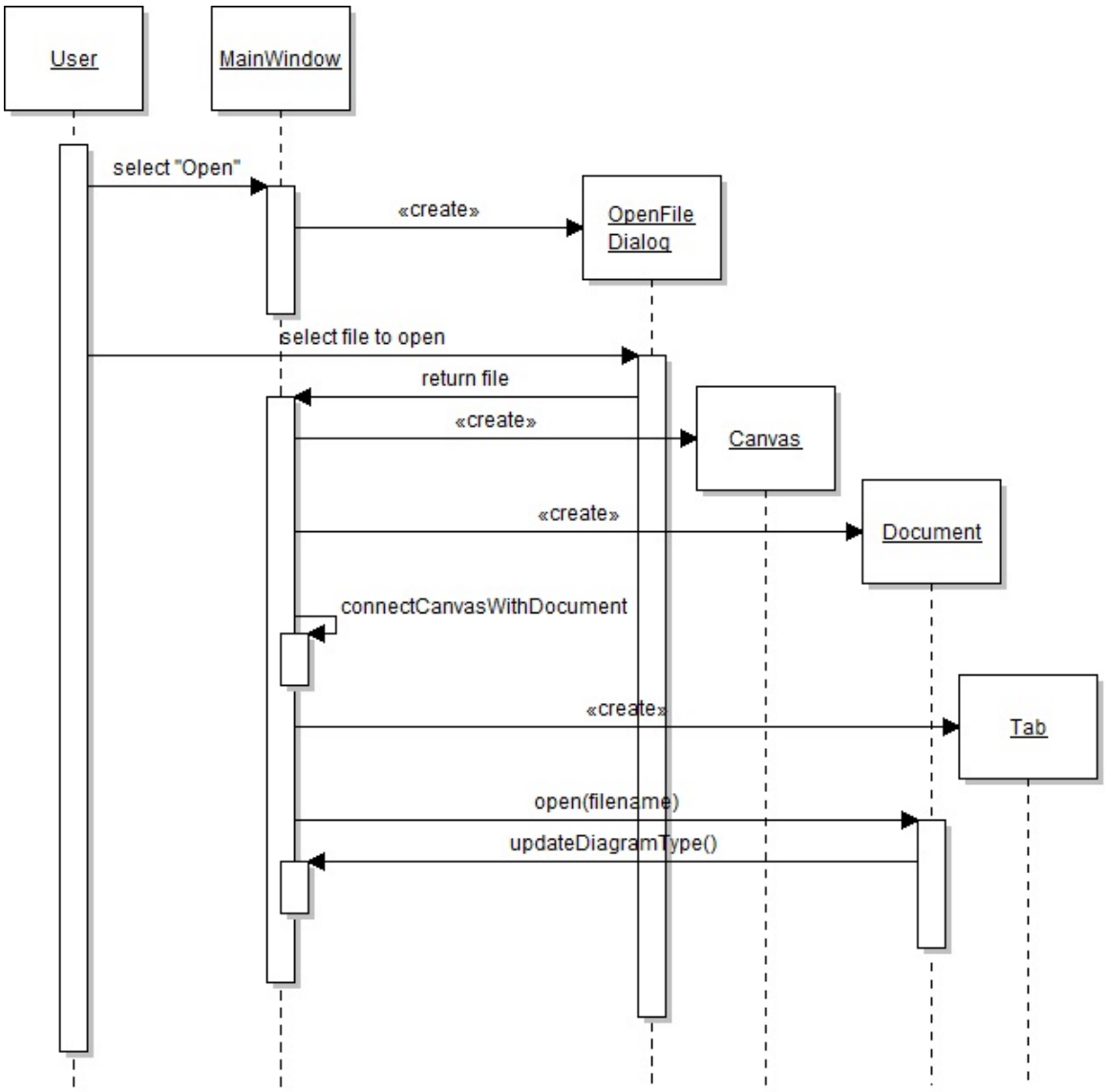


Figure 2: Open Diagram

4.3.3 Detailed Design for Feature: Exit pUML

4.3.3.1 Introduction/Purpose of this Feature When the user is finished modifying UML diagrams, the pUML software may be closed.

4.3.3.2 Input for this Feature The user clicks the "X" in the upper corner of the pUML main window, or selects "Exit" from the main menu.

4.3.3.3 Output for this Feature pUML checks to see if the file(s) being closed are saved, and once all files have been either saved or discarded, the pUML software closes.

4.3.3.4 Feature Process to Convert Input to Output User selects "Exit" from the main menu. MainWindow iterates through each of the tabs and sends a request called getModified() from the Document class. As long as none of the tabs is flagged as having a modification since the previous save, then all tabs are assumed to be saved, and MainWindow closes the pUML software.
If tabs need to be saved, reference the Save feature in this document. Tabs will be saved, and this process will resume as normal from that point.

4.3.3.5 Design Constraints and Performance Requirements of this Feature pUML will also close if diagrams are not saved, and the user wishes to discard all changes.

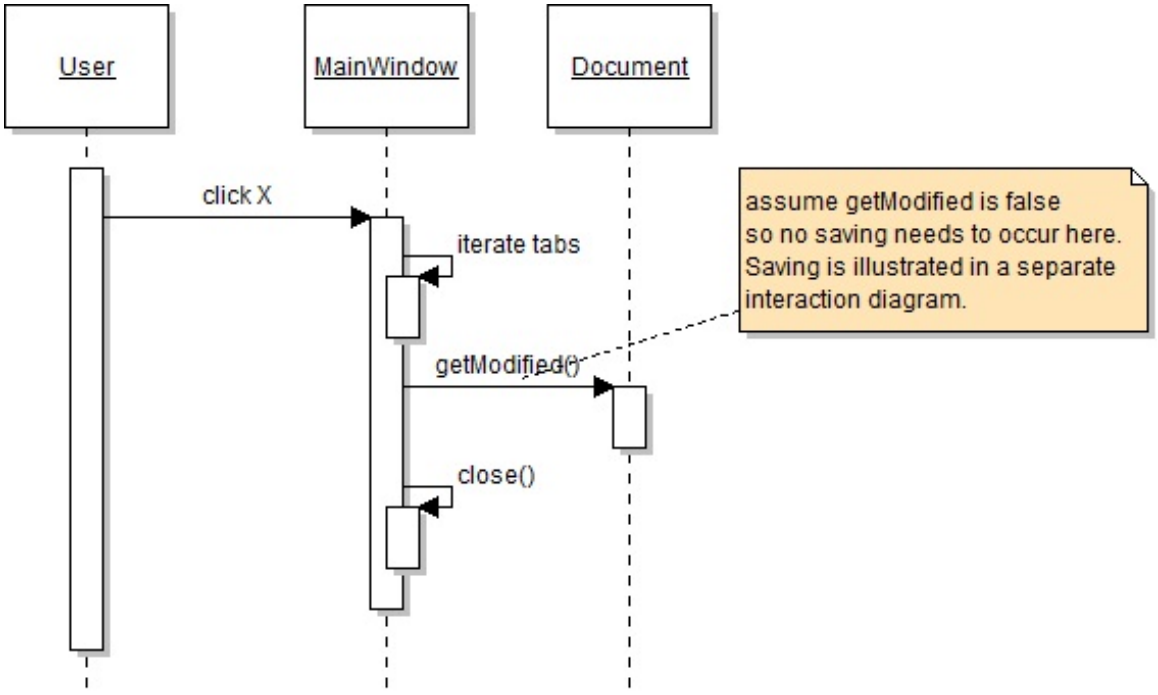


Figure 3: Exit pUML

4.3.4 Detailed Design for Feature: Save

4.3.4.1 Introduction/Purpose of this Feature The user wishes to save a UML diagram.

4.3.4.2 Input for this Feature The user opens the File Menu and selects Save.

4.3.4.3 Output for this Feature pUML checks to see if the file name that the user selects is valid, and when this is verfied, the diagram currently loaded is stored at that file location.
If the file name has not previously been stored, refer to the Save As feature in this document.

4.3.4.4 Feature Process to Convert Input to Output The user clicks "Save" on the main menu, and MainWindow sends the current diagram name through hasFileName() to Document to verify whether or not the name has been previously stored. Document verifies that hasFileName() returns true, stores the current diagram content over the contents of the existing file, and sets modifyChanged() to hide the asterisk (indicating an unsaved diagram) at that tab.

4.3.4.5 Design Constraints and Performance Requirements of this Feature If hasFileName returns false, refer to the Save As feature in this document.

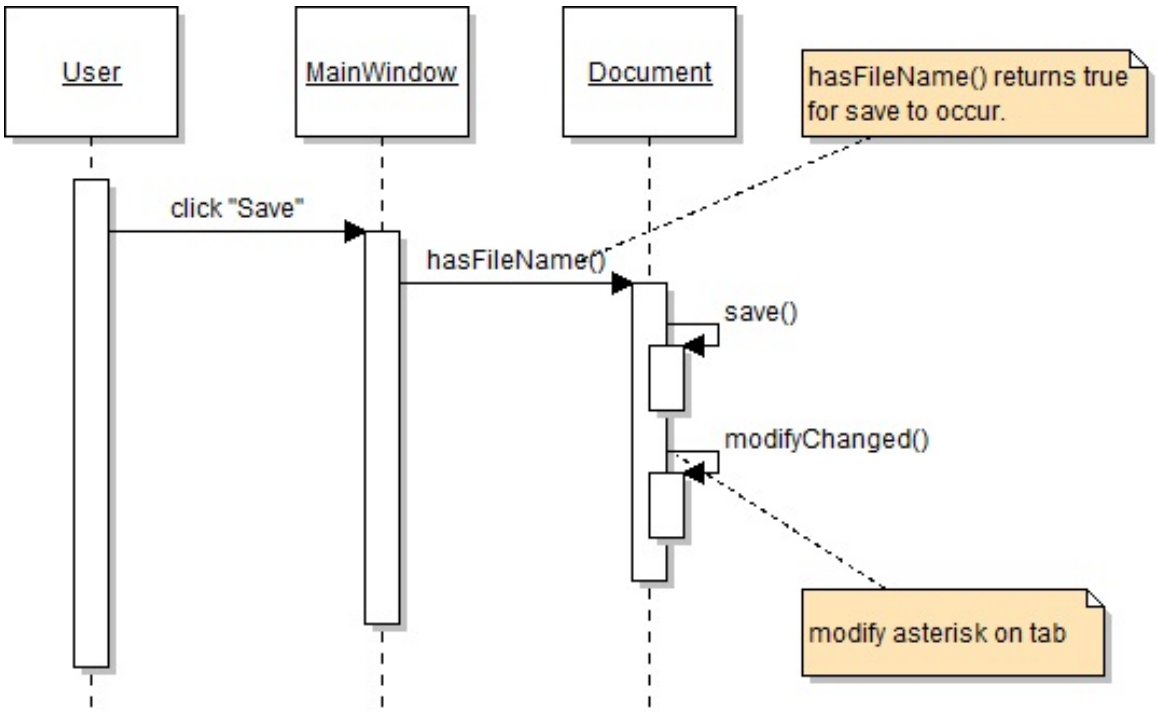


Figure 4: Save Diagram

4.3.5 Detailed Design for Feature: Save As

4.3.5.1 Introduction/Purpose of this Feature The user wishes to save a UML diagram under a new file name.

4.3.5.2 Input for this Feature The user selects Save As from the main menu.

4.3.5.3 Output for this Feature pUML produces a dialog box through which the user may type a new file name, and then pUML saves the file under that new name.

4.3.5.4 Feature Process to Convert Input to Output The User clicks on "Save As" in the main menu.
Or the User attempts to save a previously unsaved file, and MainWindow sends hasFileName() to Document to verify if the current diagram name exists, which returns false.
MainWindow creates a new QFileDialog into which the User enters their desired file name. The QFileDialog sends this information to MainWindow, which in turn calls setFileName() to Document, and then save().

4.3.5.5 Design Constraints and Performance Requirements of this Feature N/A

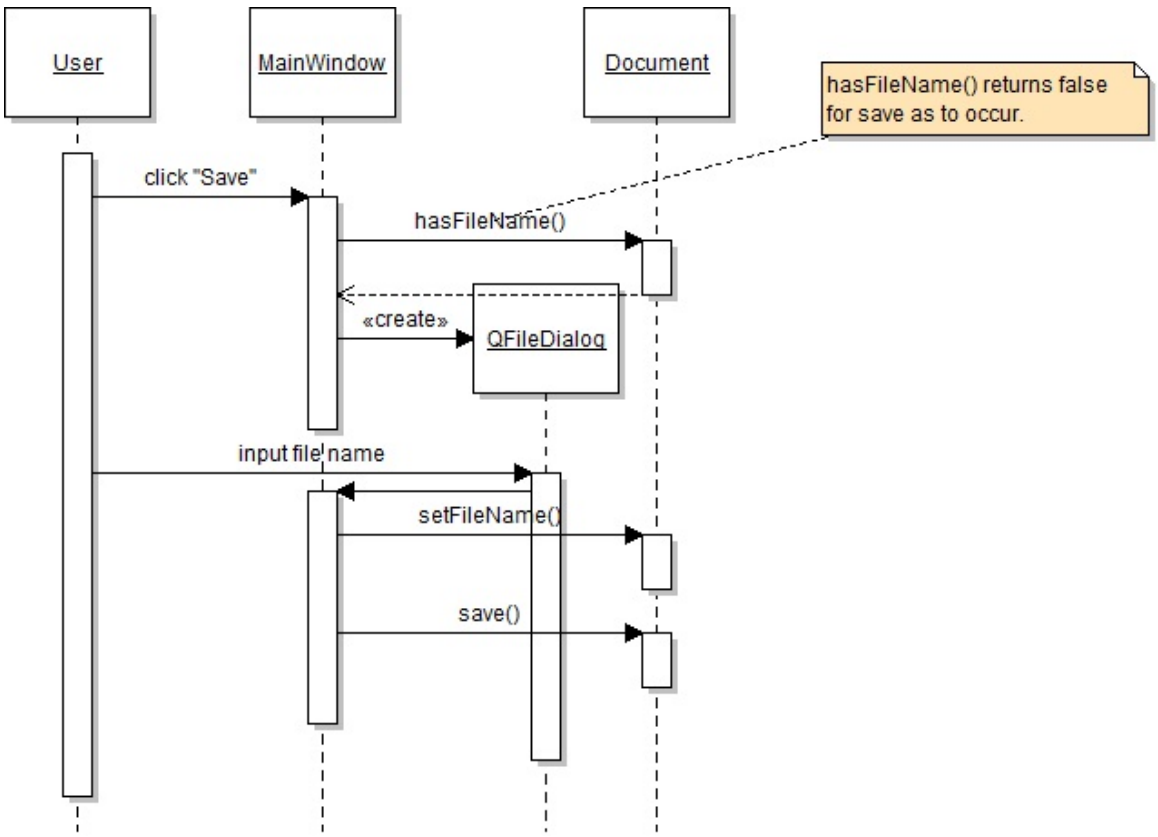


Figure 5: Save / Save As

4.3.6 Detailed Design for Feature: Close Diagram Tab

4.3.6.1 Introduction/Purpose of this Feature The pUML software provides ability for multiple diagrams to be edited by utilizing a tabbed structure. As design complexity increases, or if the User may desire to reduce screen clutter, the ability to close tabs is very useful.

4.3.6.2 Input for this Feature The user clicks the "x" on the tab.

4.3.6.3 Output for this Feature pUML checks to see if the current diagram has been saved, and when there are no outstanding modifications, the tab is closed.

4.3.6.4 Feature Process to Convert Input to Output Uesr clicks the "x" on the tabWidget, which uses MainWindow to send a getModified request to Document. If getModified returns false (diagram has been previously saved), then Document returns this information to MainWindow, which sends permission to the tabWidget to close itself.

4.3.6.5 Design Constraints and Performance Requirements of this Feature After all tabs in pUML have been closed, the pUML software remains open, with main menu options available. If the user is desiring to close the pUML software, then they must choose to "Exit" the program.

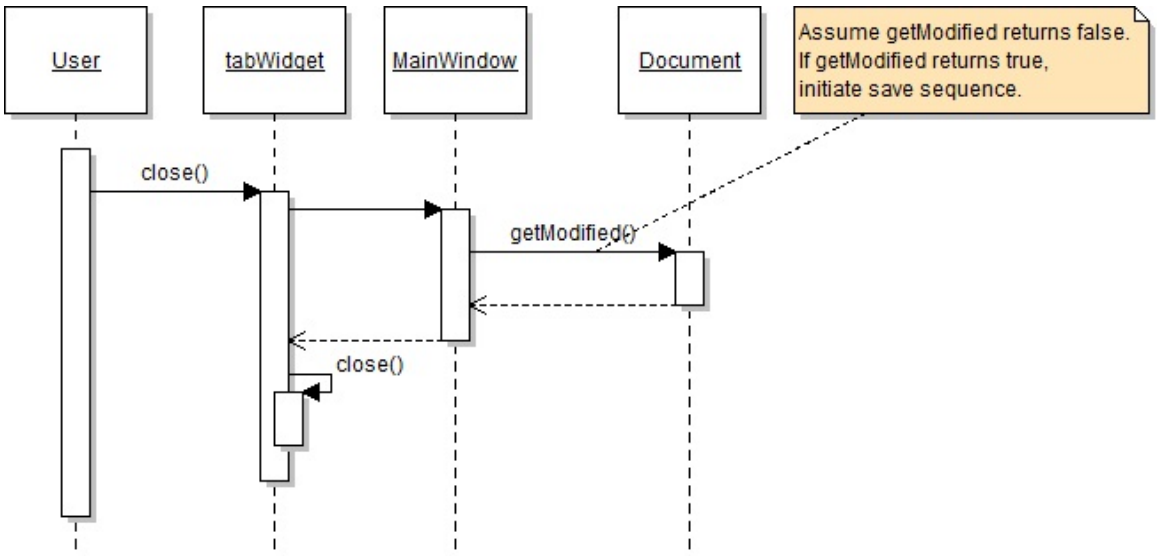


Figure 6: Close Diagram Tab

4.3.7 Detailed Design for Feature: Place New Object

4.3.7.1 Introduction/Purpose of this Feature The user will be able to select objects from the toolbar and place them on the Canvas.

4.3.7.2 Input for this Feature The user will click on a shape on the toolbar. The first subsequent click of the mouse over the Canvas area will place the selected shape at that location.

4.3.7.3 Output for this Feature The object will be placed on the Canvas, at a location of the user's choosing.

4.3.7.4 Feature Process to Convert Input to Output The mouse click on a shape on the toolbar notifies MainWindow of an objectAction(). The MainWindow sets the draw mode to object through the Canvas. The user then clicks on the canvas, which Canvas then tells Document to create the object. Document creates a new BaseNode, sends it the properties dialog, and then Document initiates showDialog().

4.3.7.5 Design Constraints and Performance Requirements of this Feature N/A

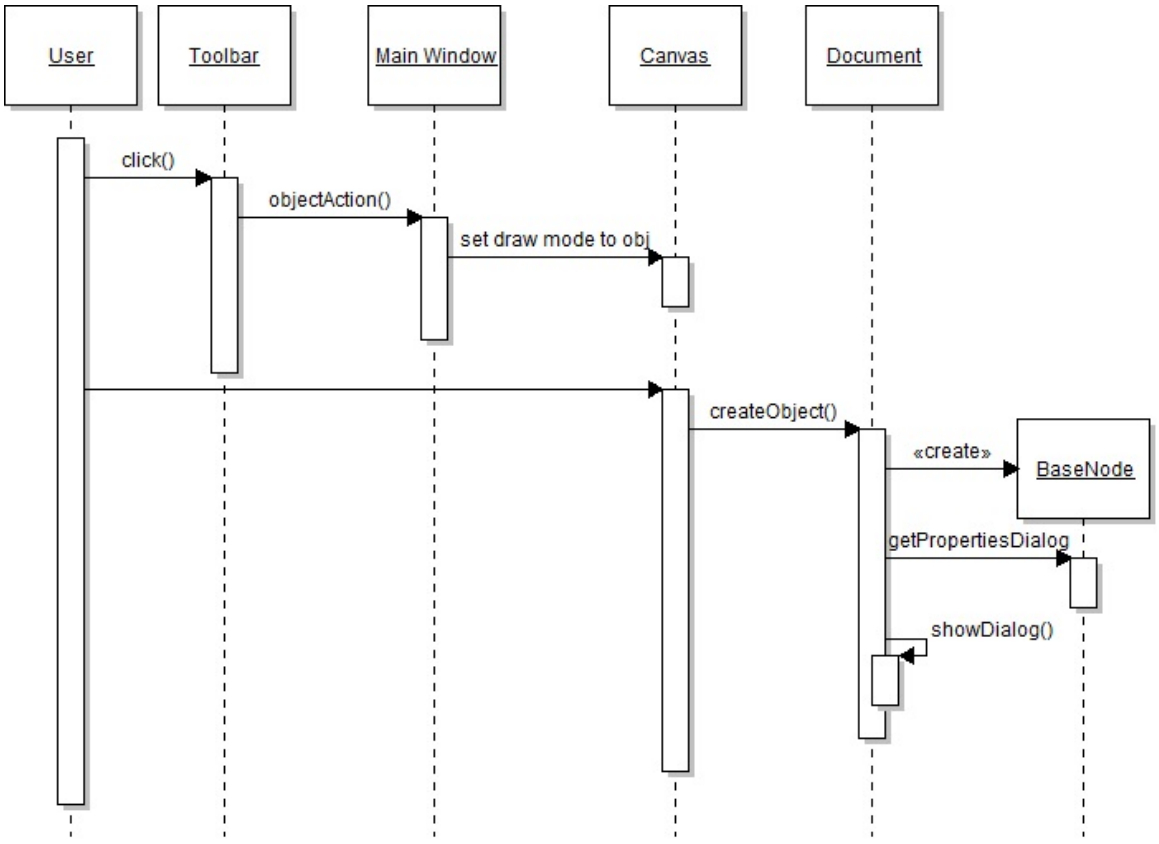


Figure 7: Place New Object

4.3.8 Detailed Design for Feature: Move Object

4.3.8.1 Introduction/Purpose of this Feature This gives the user the capability to rearrange objects in a UML diagram after their initial placement.

4.3.8.2 Input for this Feature The user clicks on an object, and then drags it to another location on the canvas.

4.3.8.3 Output for this Feature The canvas displays the object at its new location.

4.3.8.4 Feature Process to Convert Input to Output The user leftclicks down on the object, on the canvas. The Canvas class tells the Document class to setSelectedObject() to true. The user moves the mouse (mouseMove event), and Canvas redraws the object at each location. Document sets the position at each move. The user releases the mouse and the Document class retains all current object location properties.

4.3.8.5 Design Constraints and Performance Requirements of this Feature Objects are permitted to overlap in pUML, which may require more diligence on the user's part.

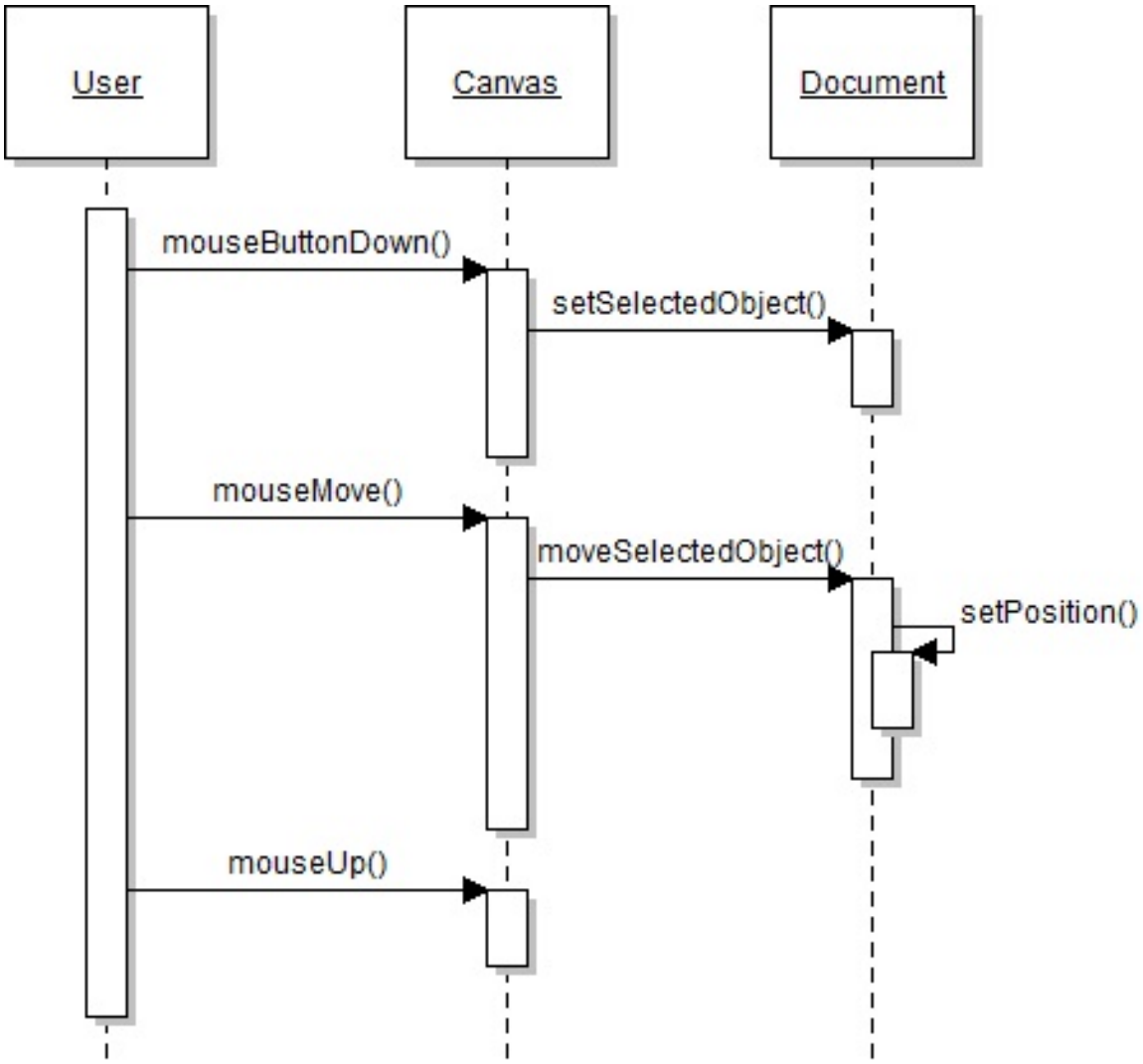


Figure 8: Move Object

4.3.9 Detailed Design for Feature: Edit Object Description

4.3.9.1 Introduction/Purpose of this Feature The user may edit an object's description.

4.3.9.2 Input for this Feature The user right clicks on the object they wish to modify and selects "Properties" from the context menu.

4.3.9.3 Output for this Feature pUML displays a dialog box with object description, and after the user has submitted changes to the description, the new description is displayed on the canvas.

4.3.9.4 Feature Process to Convert Input to Output User right clicks on an object on the Canvas. The Canvas tells Document to perform a hittest to determine whether an object resides at those coordinates. If valid, the Canvas tells Document to initiate the properties Dialog for the selected object.
The user modifies the object properties per the Dialog. The Dialog sends the updated object information to Document.

4.3.9.5 Design Constraints and Performance Requirements of this Feature The user may only edit properties of valid objects and connectors. No other properties in pUML may be modified.

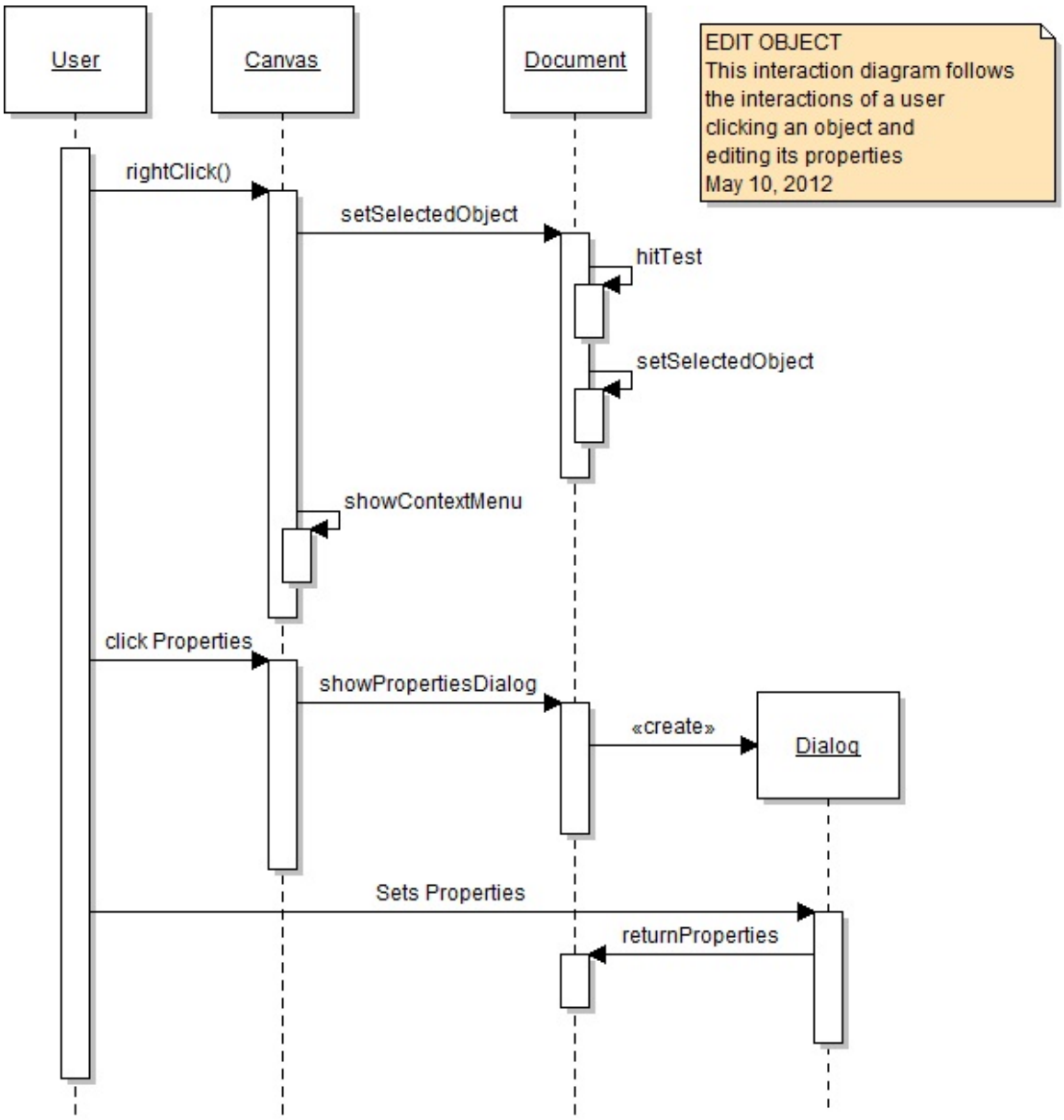


Figure 9: Edit Object Description

4.3.10 Detailed Design for Feature: Delete an Object

4.3.10.1 Introduction/Purpose of this Feature The user deletes an object from the Canvas.

4.3.10.2 Input for this Feature The user right-clicks on an object to be deleted, and selects the delete option from the context menu.

4.3.10.3 Output for this Feature The object is removed from the canvas along with any connectors (connected objects).

4.3.10.4 Feature Process to Convert Input to Output User right clicks on an object on the Canvas. The Canvas verifies that an object exists at the coordinates of the click, and sets the object through Document. The Canvas displays the context menu. The user selects the delete option from the menu, which tells Document to delete the object and any connected objects.

4.3.10.5 Design Constraints and Performance Requirements of this Feature All connectors attached to an object will be deleted along with that object.

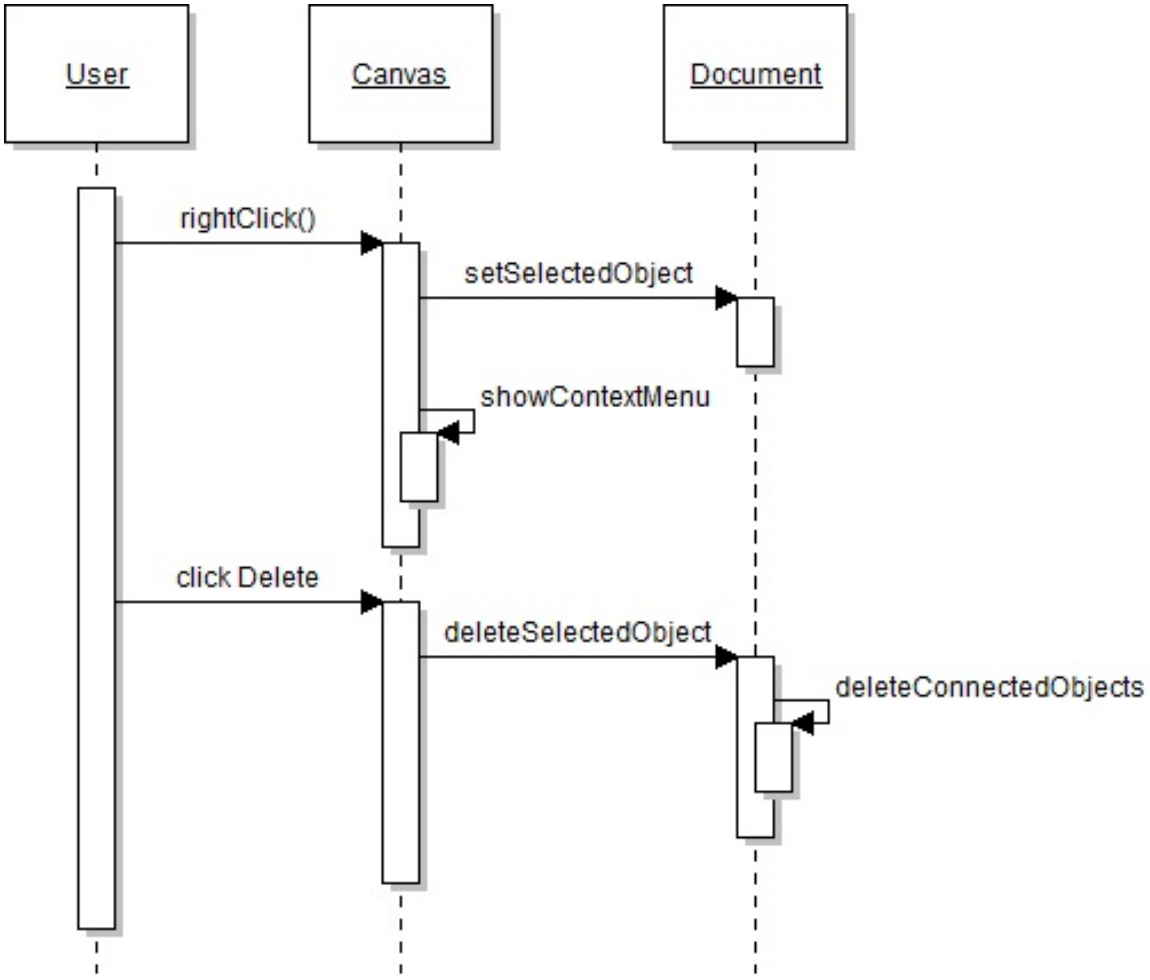


Figure 10: Delete Object

4.3.11 Detailed Design for Feature: Place a New Connector

4.3.11.1 Introduction/Purpose of this Feature This feature allows the user to place a connecting line between two objects.

4.3.11.2 Input for this Feature The user selects a connector shape from the toolbar, and then selects two objects between which to place the connector.

4.3.11.3 Output for this Feature A connector line is drawn between the two objects the user selected.

4.3.11.4 Feature Process to Convert Input to Output User clicks on a connector shape in the Toolbar, which sends a connectorAction to the Main Window. MainWindow tells the Canvas to set the draw mode to connector.
The user clicks and holds down the mouse, presumably over an object. The Canvas tells Document to create the first Connection Point at that location. Document performs a hittest to ensure that a valid object node resides at that location.
The user releases the mousebutton, which Canvas sends to Document as the second connection point. After performing a hittest, Document creates a BaseNode, tells it to addConnectedNode at each connection point. Document addsConnectedNode after this and the connection line is drawn.

4.3.11.5 Design Constraints and Performance Requirements of this Feature As of this release, it is possible to create several identical connection lines between two objects, and so users must proceed with caution.

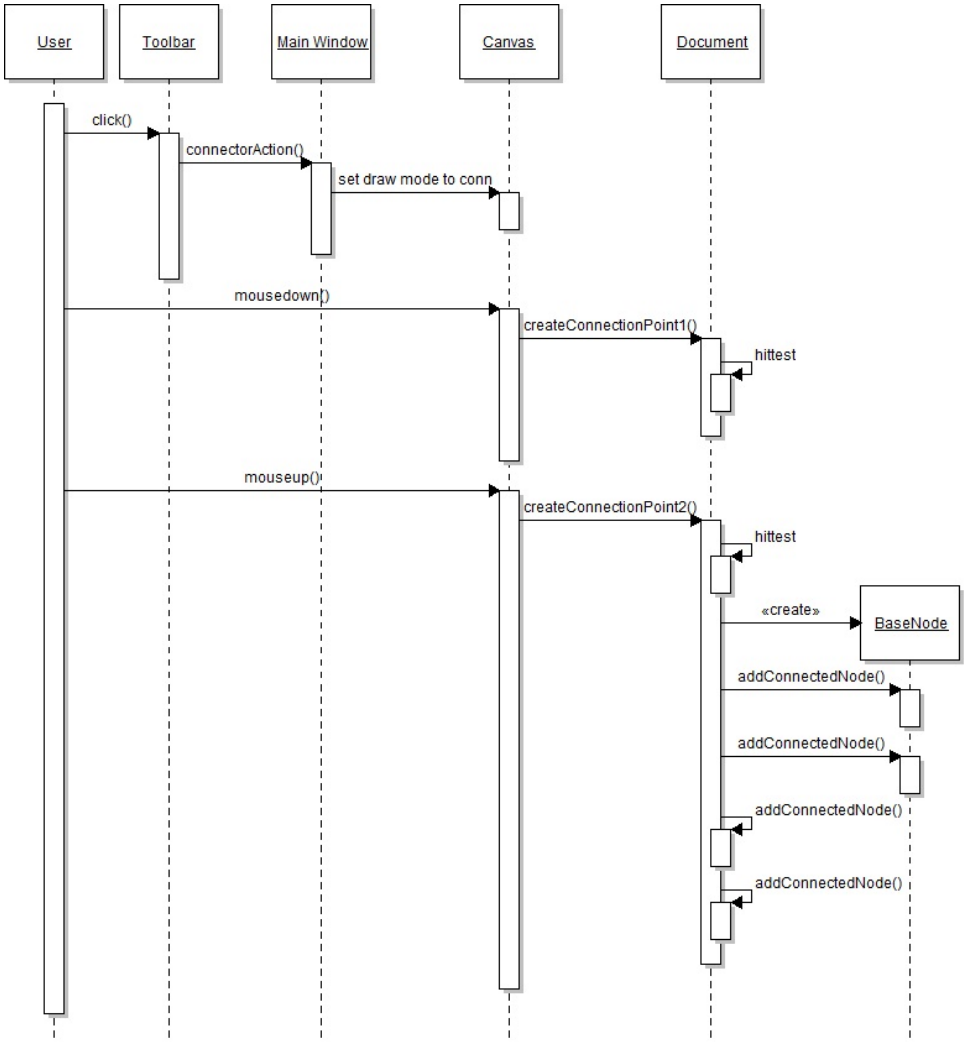


Figure 11: Place New Connector

4.3.12 Detailed Design for Feature: Edit Connector Description

4.3.12.1 Introduction/Purpose of this Feature User may edit a connector's description.

4.3.12.2 Input for this Feature The user right clicks on a connector and selects "Properties" from the context menu.

4.3.12.3 Output for this Feature A dialog is displayed from which the user may make changes to the connector's description. After making a change, the new description will replace the existing description.

4.3.12.4 Feature Process to Convert Input to Output User right-clicks on the canvas. Canvas class setsSelectedConnector() to Document, which performs a hittest to verify the connector at the coordinates of the click. Then Document setSelectedConnector to itself.
Canvas displays the context menu. The User clicks "Properties" and Canvas sends a showPropertiesDialog signal to Document, which in turn creates a properties Dialog.
The User then modifies the properties, and Dialog sends the new properties information to Document.

4.3.12.5 Design Constraints and Performance Requirements of this Feature The user may not be able to edit all properties of all connectors due to the differences in connector utilization in differing UML diagrams.
The user may only edit properties of valid objects and connectors. No other properties in pUML may be modified.

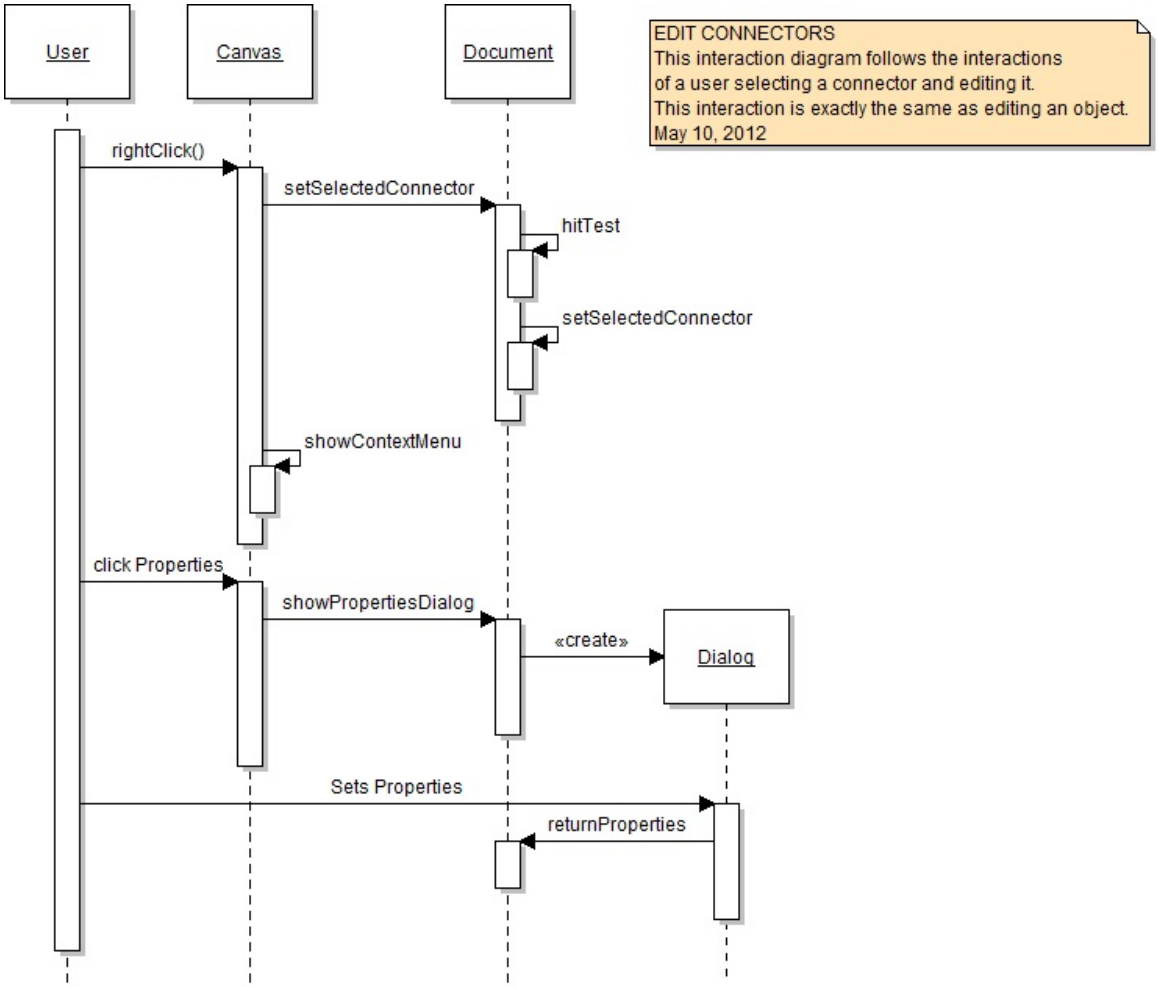


Figure 12: Edit Connector Description

4.3.13 Detailed Design for Feature: Delete Connector

4.3.13.1 Introduction/Purpose of this Feature The user deletes a connector from between two objects.

4.3.13.2 Input for this Feature The user right-clicks on an connector, and selects "Delete" from the context menu.

4.3.13.3 Output for this Feature The connector no longer appears on the canvas.

4.3.13.4 Feature Process to Convert Input to Output User right clicks on an connector on the Canvas. The Canvas verifies through Document that a connector exists at the coordinates of the click, and sets the connector. The Canvas displays a context menu. The user selects the "Delete" option from the context menu. Canvas tells Document to deleteSelectedConnector, and Document deletes the connector.

4.3.13.5 Design Constraints and Performance Requirements of this Feature N/A

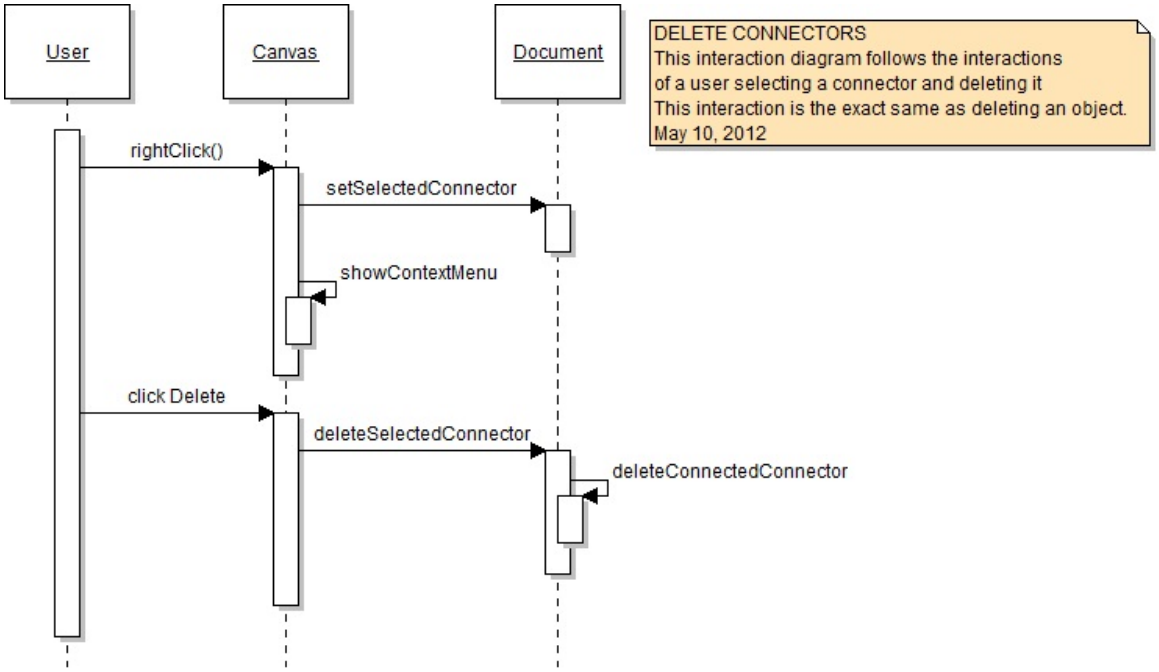


Figure 13: Delete Connector

5 REQUIREMENTS TRACEABILITY

Feature Name	Req No.	Requirement Description	Priority	SSDD	Test Case(s)	Alpha Release	Beta Release	Final Test
Install	2.2.1	pUML software installs successfully	M	N/A	testinstall	PASS	PASS	PASS
Launch	2.2.1	pUML software launches successfully	M	N/A	testlaunch	PASS	PASS	PASS
Exit	2.2.1	pUML Exit options	M	4.3.3	testsave * testsaveas *	FAIL FAIL	PASS PASS	PASSPASS
Main Window	2.2.2	All main window features	M	*N/A	testmainwindow	PASS	PASS	
Objects	2.2.3	Object properties and functionality	M	4.3.7 4.3.8 4.3.9 4.3.10	testobjects	PASS	PASS	
Connectors	2.2.4	Connector properties and functionality	M	4.3.11 4.3.12 4.3.13	testconnectors testselfconnector	FAIL PASS	PASS PASS	PASS
Open	2.2.5	Load a pUML file into pUML	M	4.3.2	testsave * testsaveas *	FAIL FAIL	PASS PASS	
Save/Save As	2.2.6	Save a diagram	M	4.3.4 4.3.5	testsave testsaveas	FAIL PASS	PASS PASS	
Diagrams	2.2.7	Diagram type integrity	M	4.3.1	testsave * testsaveas *	FAIL FAIL	PASS PASS	
Tabs	2.2.8	Tab functionality	M	4.3.6	testsave * testsaveas *	FAIL FAIL	PASS PASS	

Priorities are: **M**andatory, **L**ow, **H**igh
SSDD link is version and page number or function name.
Test cases and results are file names and **P**ass/**F**ail or % passing.
* asterisk indicates indirectly tested through another feature test case.