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

Phunctional UML Editor (pUML)

Version 1.0
May 8, 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, class and interaction diagrams	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

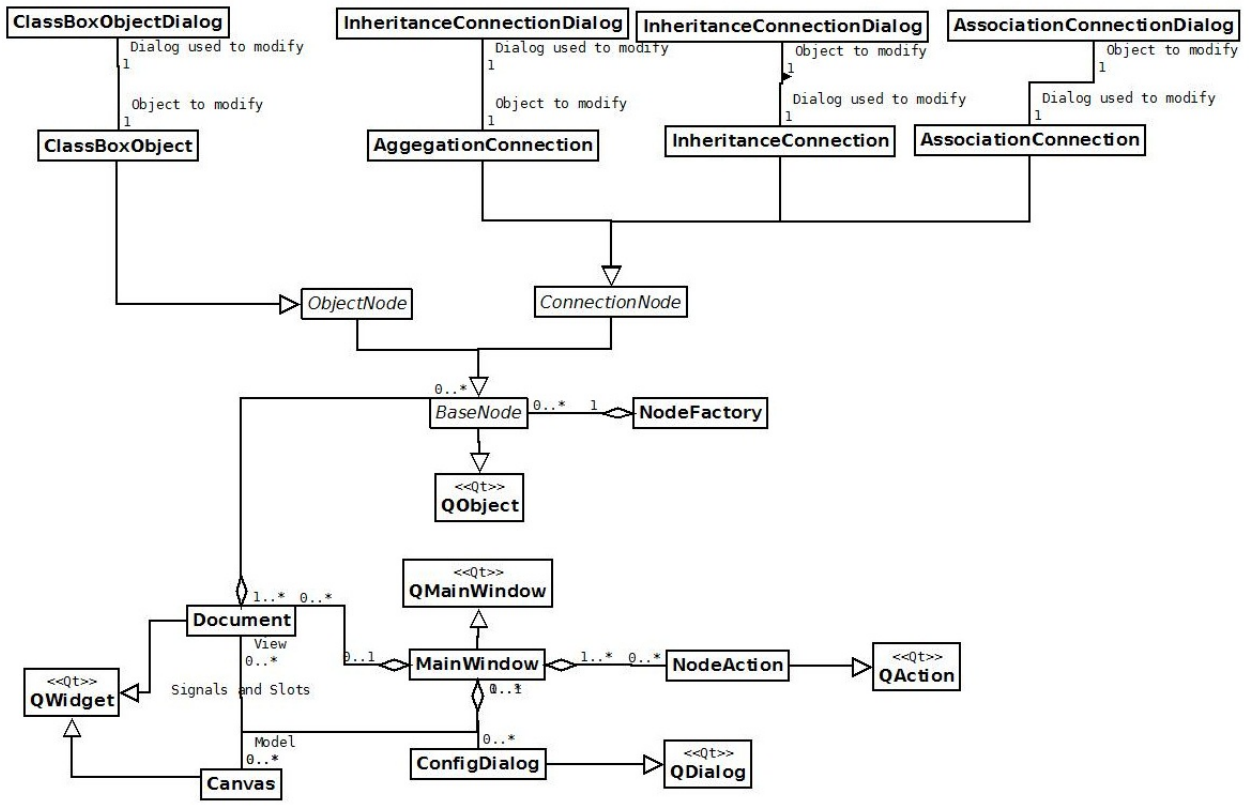
Due to the limited scope of this project, there are no known inconsistencies between views.

4 SOFTWARE DETAILED DESIGN

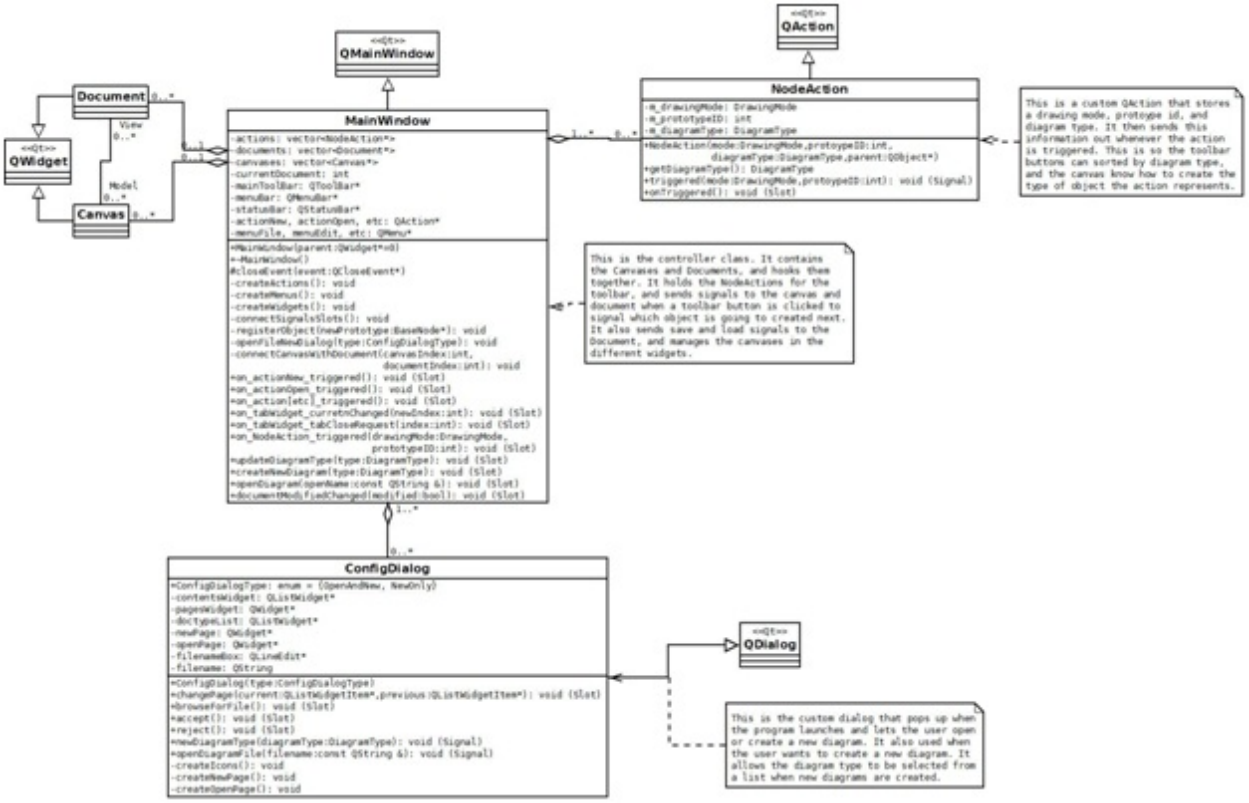
4.1 DEVELOPER'S VIEWPOINT DETAILED SOFTWARE DESIGN

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

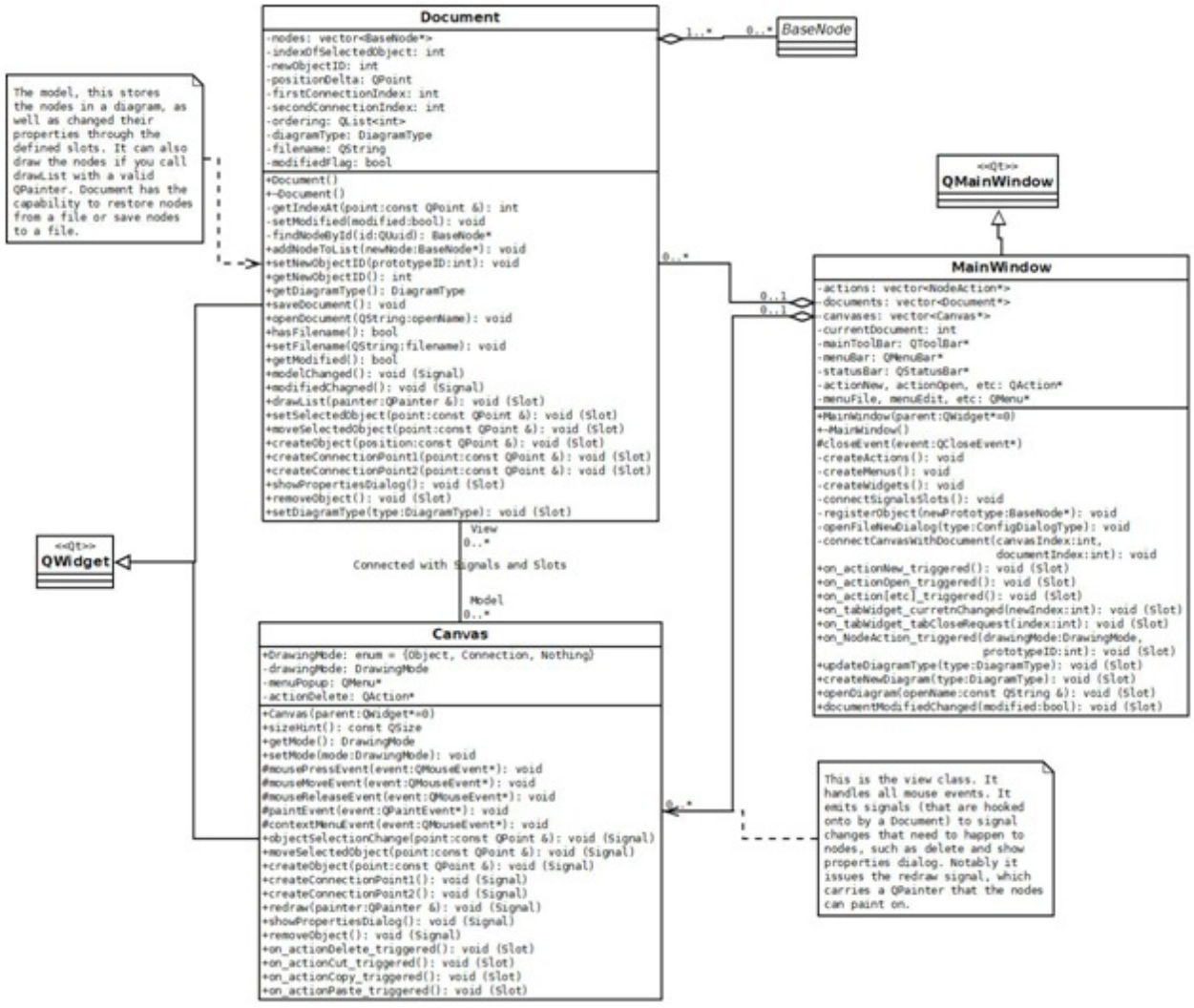
4.1.1 Class Overview



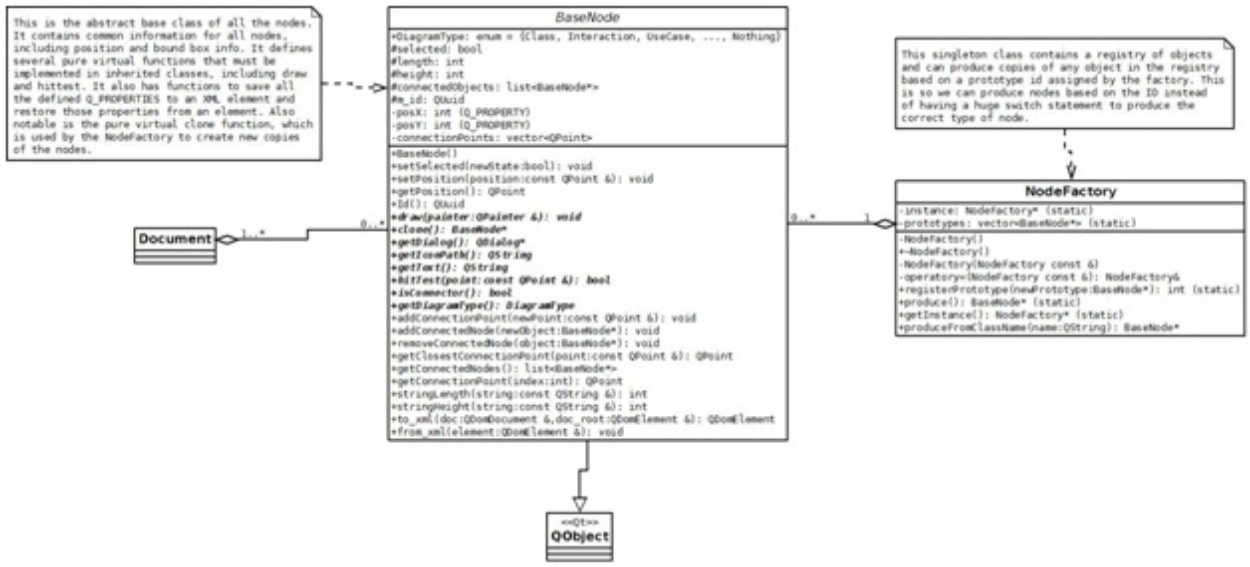
4.1.2 Main Window Class



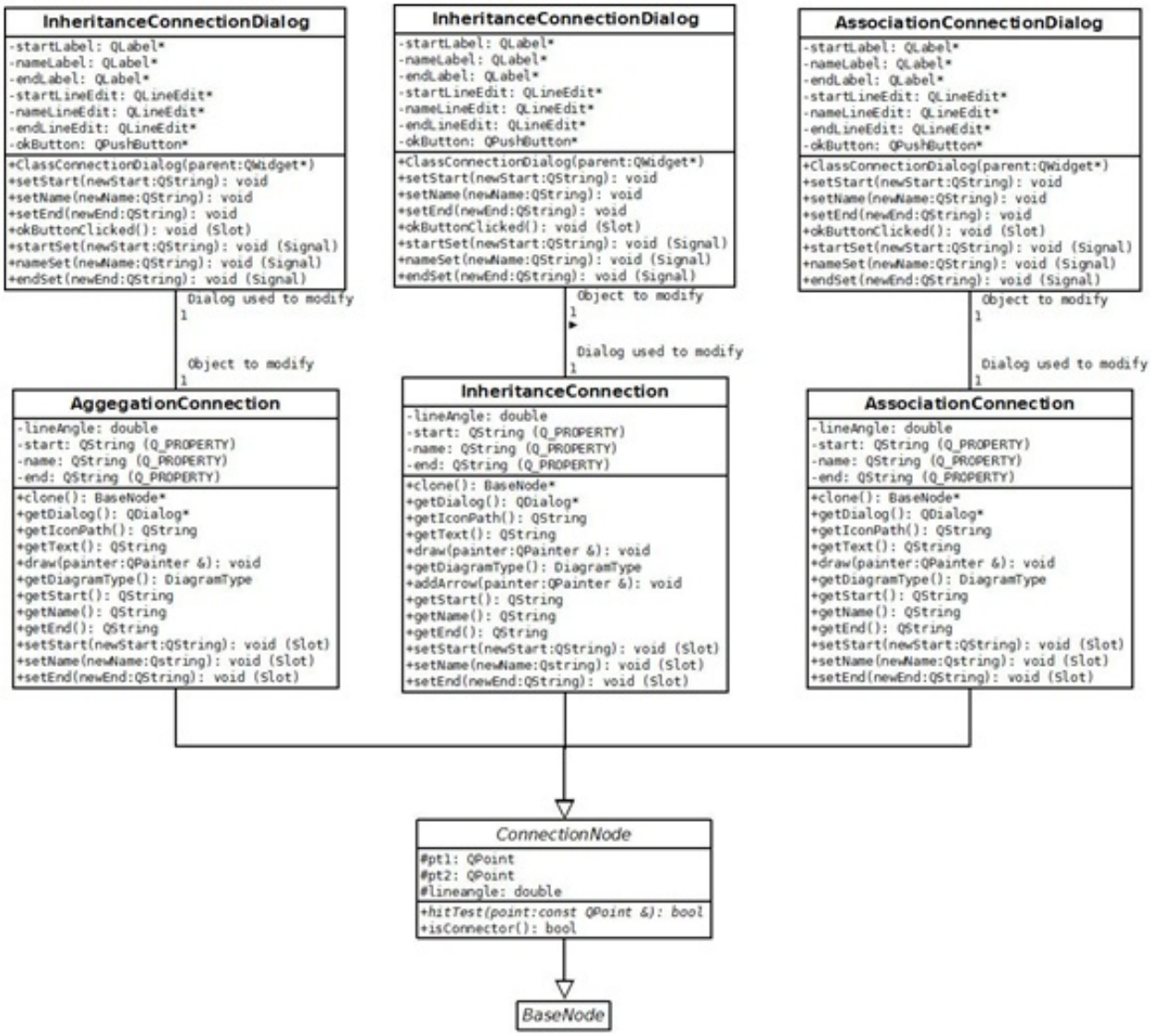
4.1.3 Document and Canvas Classes



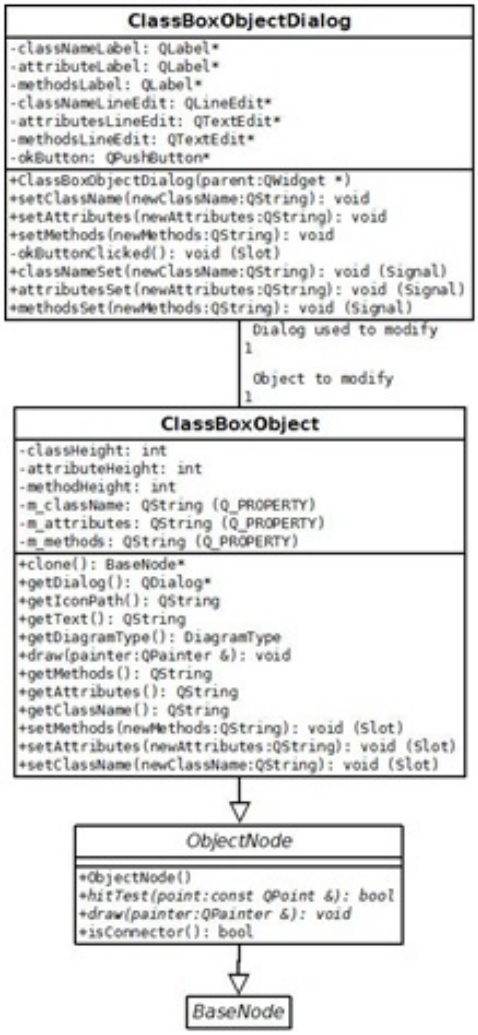
4.1.4 Base Node Class



4.1.5 Connnection Node Class



4.1.6 Object Node Class



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

4.3.1.2 Input for this Feature

4.3.1.3 Output for this Feature

4.3.1.4 Feature Process to Convert Input to Output

Figure 1: Create New Diagram

4.3.1.5 Design Constraints and Performance Requirements of this Feature

4.3.2 Detailed Design for Feature: Open an Existing Diagram

4.3.2.1 Introduction/Purpose of this Feature

4.3.2.2 Input for this Feature

4.3.2.3 Output for this Feature

4.3.2.4 Feature Process to Convert Input to Output

Figure 2: Open Diagram

4.3.2.5 Design Constraints and Performance Requirements of this Feature

4.3.3 Detailed Design for Feature: Exit pUML

4.3.3.1 Introduction/Purpose of this Feature

4.3.3.2 Input for this Feature

4.3.3.3 Output for this Feature

4.3.3.4 Feature Process to Convert Input to Output

Figure 3: Exit pUML

4.3.3.5 Design Constraints and Performance Requirements of this Feature

4.3.4 Detailed Design for Feature: Save As

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

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

4.3.4.3 Output for this Feature pUML produces a QDialog box through which the user may type a new file name, and then saves the file.

4.3.4.4 Feature Process to Convert Input to Output User clicks on the File menu option. User clicks on the Save As option. Main Menu creates a predefined QDialog and shows it to the user. The user types in the desired file name, and the QDialog submits this information to the Main Menu. Main Menu sends updated information about the file's saved status to the Canvas. Canvas acknowledges and returns xml. Main Menu prints new xml information.

4.3.4.5 Design Constraints and Performance Requirements of this Feature Directory browsing functionality within the QDialog may not be developed until a later release.

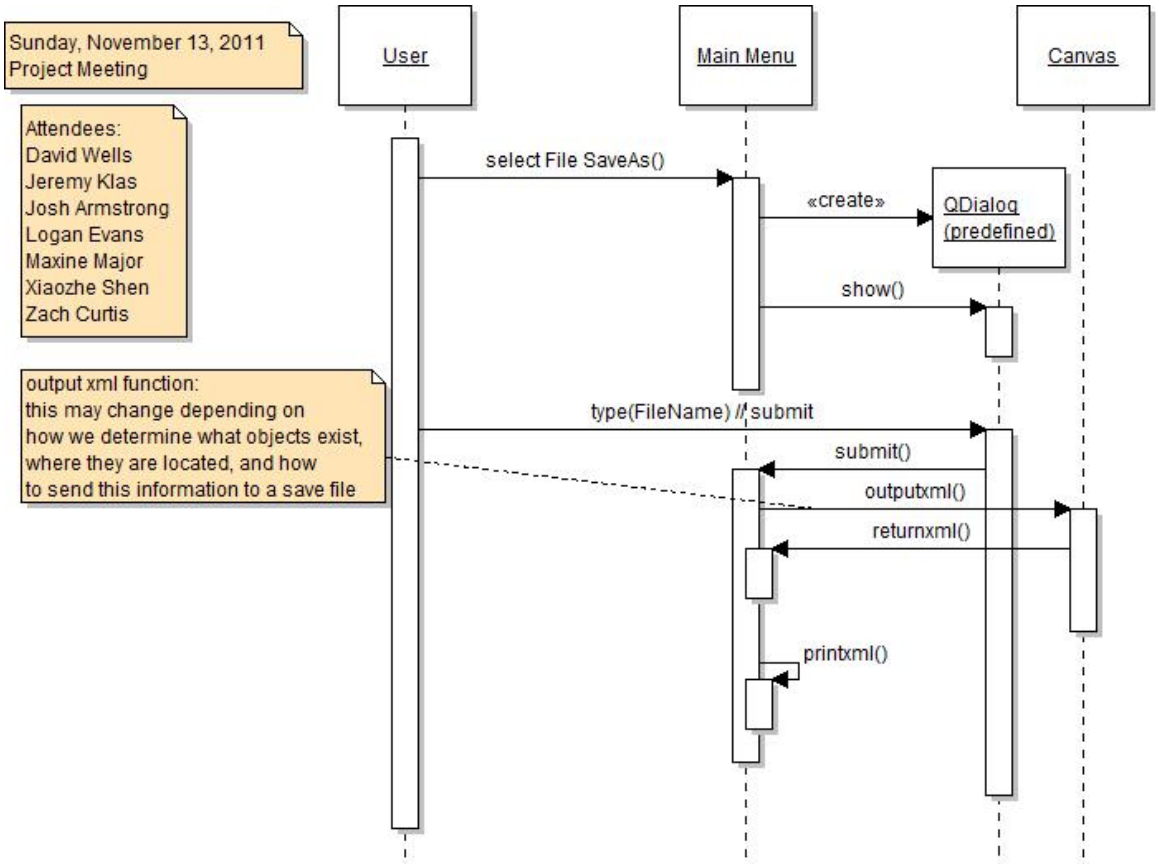


Figure 4: Save / Save As

4.3.5 Detailed Design for Feature: Close Diagram Tab

4.3.5.1 Introduction/Purpose of this Feature

4.3.5.2 Input for this Feature

4.3.5.3 Output for this Feature

4.3.5.4 Feature Process to Convert Input to Output

Figure 5: Close Diagram Tab

4.3.5.5 Design Constraints and Performance Requirements of this Feature

4.3.6 Detailed Design for Feature: Move Object

4.3.6.1 Introduction/Purpose of this Feature

4.3.6.2 Input for this Feature

4.3.6.3 Output for this Feature

4.3.6.4 Feature Process to Convert Input to Output

Figure 6: Move Object

4.3.6.5 Design Constraints and Performance Requirements of this Feature

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, provided there are no collisions with other objects.

4.3.7.4 Feature Process to Convert Input to Output The mouse click on a shape on the toolbar will lock that shape in memory. The subsequent mouse click on the Canvas will send the shape information from the toolbar to the Canvas. The Canvas will draw the shape at the specific coordinates of the mouse click, and will update the list of existing shapes on the Canvas for that particular diagram.

4.3.7.5 Design Constraints and Performance Requirements of this Feature The object will not be permitted to be placed at an overlapping location with another object.

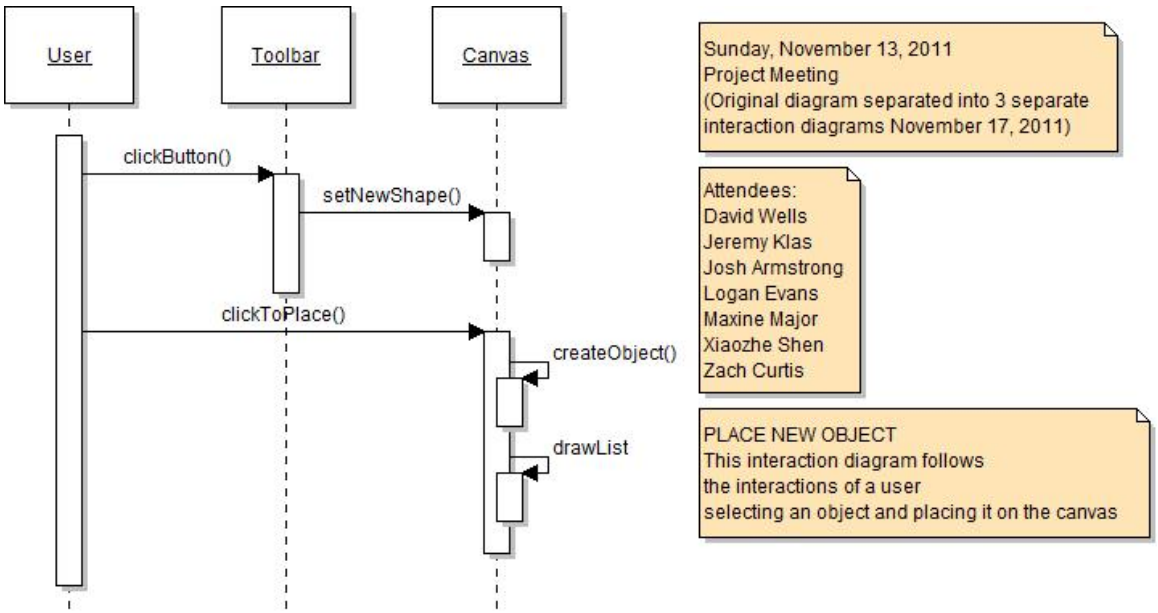


Figure 7: Place New Object

4.3.8 Detailed Design for Feature: Select an Object

4.3.8.1 Introduction/Purpose of this Feature User wishes to select an object, either for editing purposes or to move the object.

4.3.8.2 Input for this Feature User clicks on the object.

4.3.8.3 Output for this Feature The canvas identifies and displays the object as selected.

4.3.8.4 Feature Process to Convert Input to Output The user clicks on an object. Canvas stores the coordinates of the click and sends them to nodes to test whether an object resides at those coordinates on the Canvas. nodes returns true or false. Canvas updates the index of selected items, and draws this list back to nodes.

4.3.8.5 Design Constraints and Performance Requirements of this Feature Actions that may be based upon an object's selection have not been developed at this time.

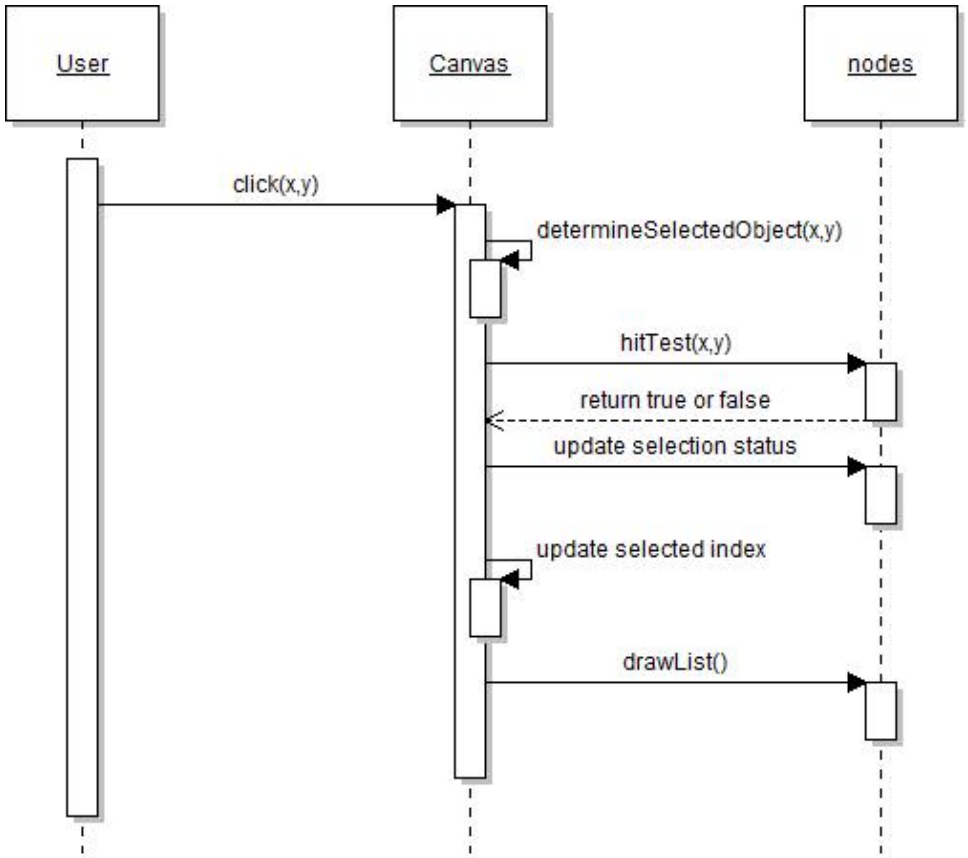


Figure 8: Select an Object

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 properties, such as names, descriptions, etc..

4.3.9.2 Input for this Feature The user right clicks on the object they wish to modify.

4.3.9.3 Output for this Feature The program displays a dialog box with object properties the user may modify, and updates the program with changes.

4.3.9.4 Feature Process to Convert Input to Output User right clicks on an object on the Canvas. The Canvas determines whether an object resides at those coordinates. If valid, the Canvas sets that object, and activates QMenu, which in turn shows the Dialog for the selected object. The user modifies the object properties per the Dialog. The Dialog sends the updated object information back to the Canvas.

4.3.9.5 Design Constraints and Performance Requirements of this Feature The user may not edit Canvas properties.

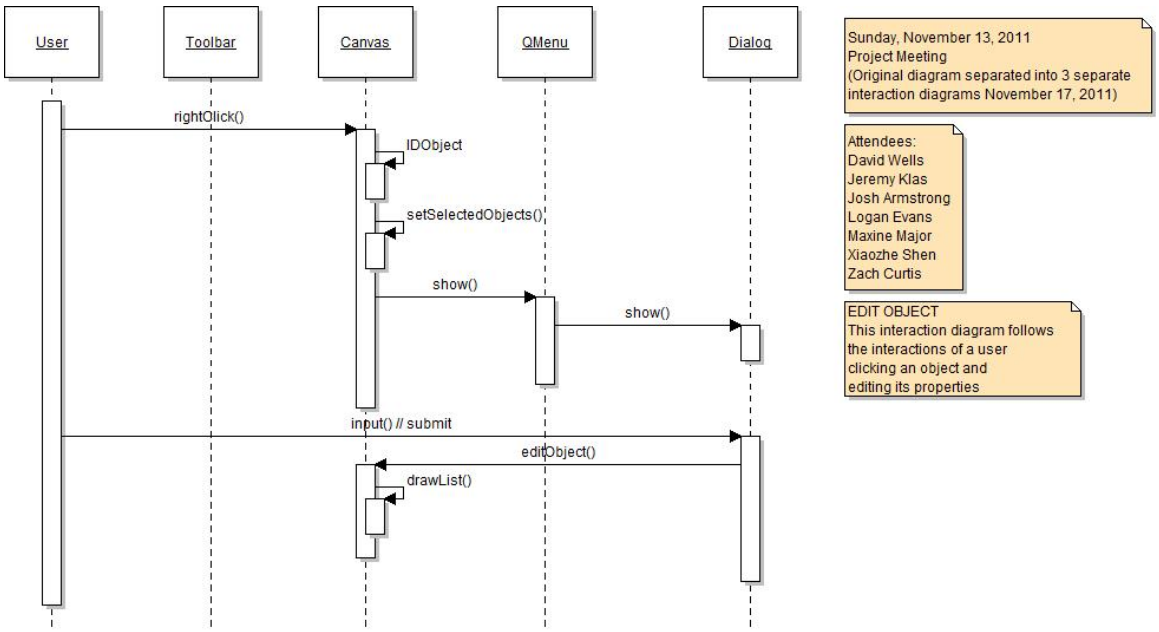


Figure 9: Edit Object Description

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 QMenu.

4.3.10.3 Output for this Feature The Canvas removes the object from itself.

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. The Canvas prompts the QMenu to show itself. The user selects the delete option from the QMenu. The QMenu tells the Canvas to delete that object. The Canvas deletes the object and draws the updated list of objects.

4.3.10.5 Design Constraints and Performance Requirements of this Feature It has not been determined at this time when an object is deleted if any associated connectors will automatically be deleted as well.

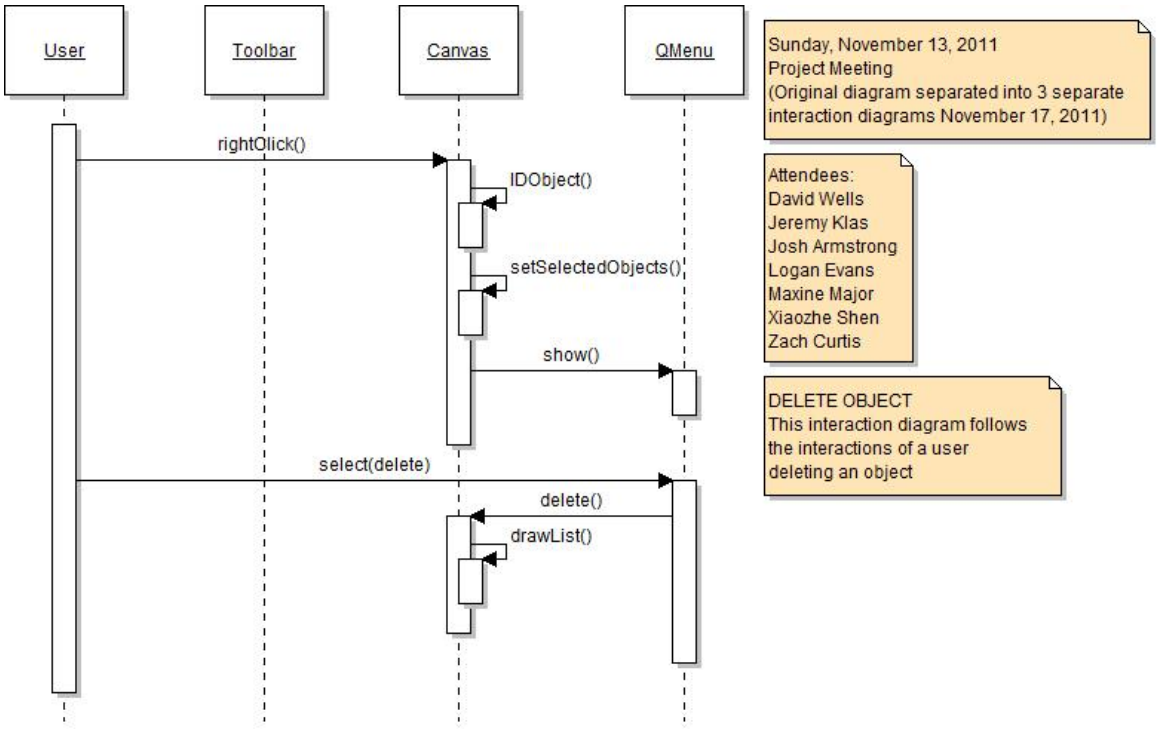


Figure 10: Delete Object

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. The Canvas gets the connector information from the Toolbar. The user clicks on an object. The Canvas verifies that an object exists at that location, identifies the closest connection point on the object to the click, and stores this information. The user clicks on a second object on the Canvas. The Canvas verifies that an object exists at the second location, identifies the closest connection point on the object to the click, and stores this information. The Canvas draws the connector at the coordinates of the user's click on the Canvas. Canvas updates list of objects on the Canvas.

4.3.11.5 Design Constraints and Performance Requirements of this Feature More than one connector may not connect the same two objects.

A connector may not be placed if two different objects have not been selected to place the connector between.

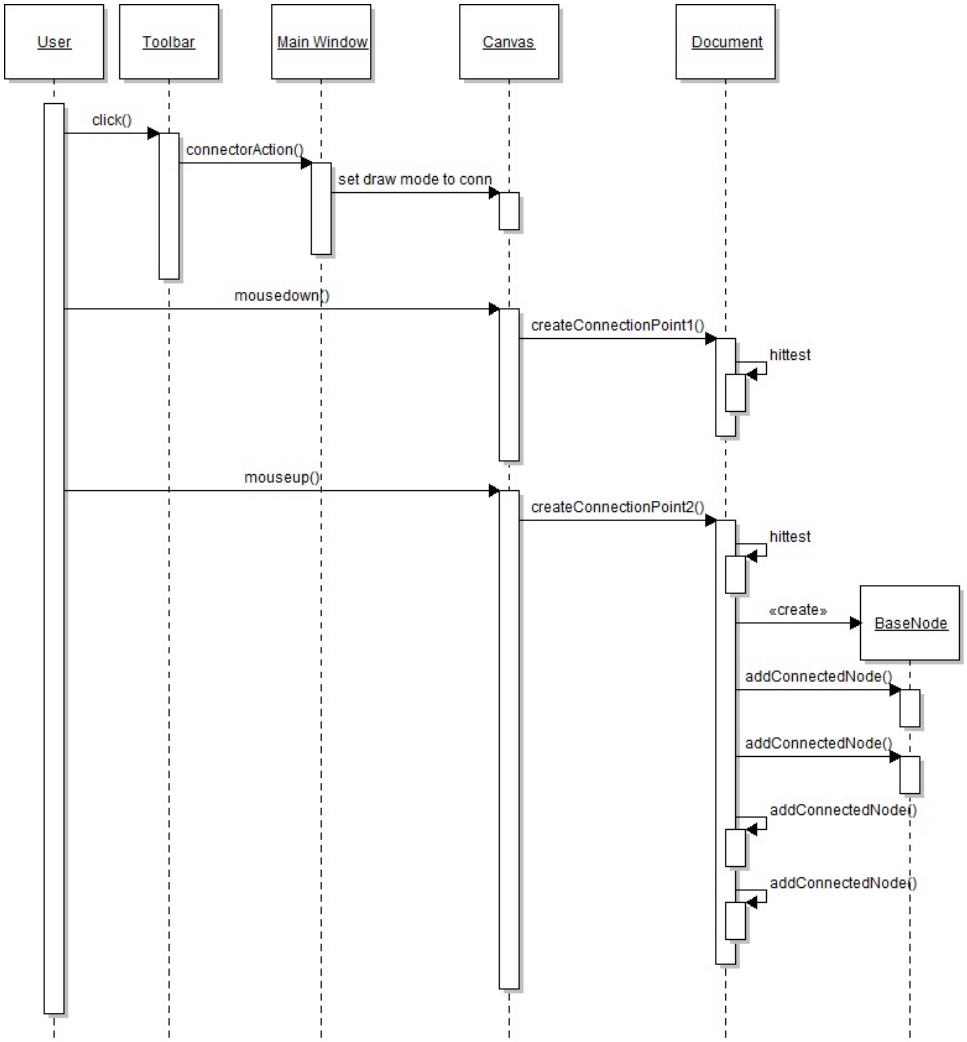


Figure 11: Place New Connector

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 features, such as name, descriptors, arrowheads, direction, etc..

4.3.12.2 Input for this Feature The user right clicks on a connector.

4.3.12.3 Output for this Feature A dialog is displayed from which the user may make changes to the connector's properties.

4.3.12.4 Feature Process to Convert Input to Output User right clicks on a connector on the Canvas. The Canvas determines whether a connector resides at those coordinates. If valid, the Canvas sets that connector, and activates QMenu, which in turn shows the Dialog for the selected connector. The user modifies the connector properties per the Dialog. The Dialog sends the updated connector information back to the Canvas.

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.

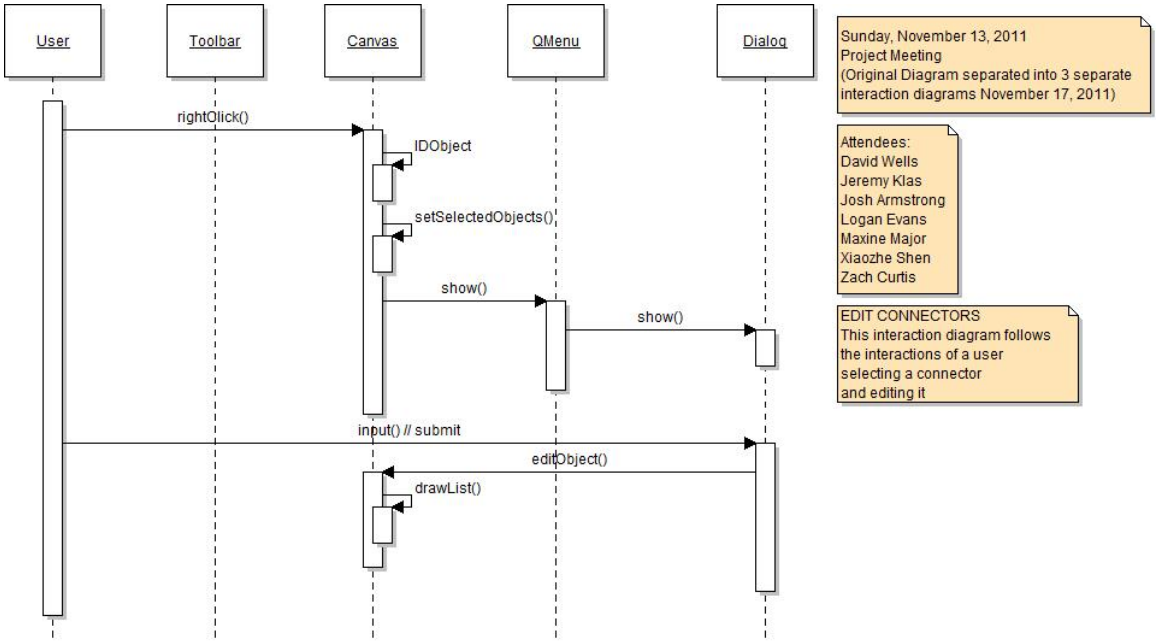


Figure 12: Edit Connector Description

4.3.13 Detailed Design for Feature: Select Connector

4.3.13.1 Introduction/Purpose of this Feature

4.3.13.2 Input for this Feature

4.3.13.3 Output for this Feature

4.3.13.4 Feature Process to Convert Input to Output

Figure 13: Select Connector

4.3.13.5 Design Constraints and Performance Requirements of this Feature

4.3.14 Detailed Design for Feature: Delete Connector

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

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

4.3.14.3 Output for this Feature The Canvas deletes the connector.

4.3.14.4 Feature Process to Convert Input to Output User right clicks on an connector on the Canvas. The Canvas verifies that an connector exists at the coordinates of the click, and sets the connector. The Canvas prompts the QMenu to show itself. The user selects the delete option from the QMenu. The QMenu tells the Canvas to delete that connector. The Canvas deletes the connector and draws the updated list of connectors.

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

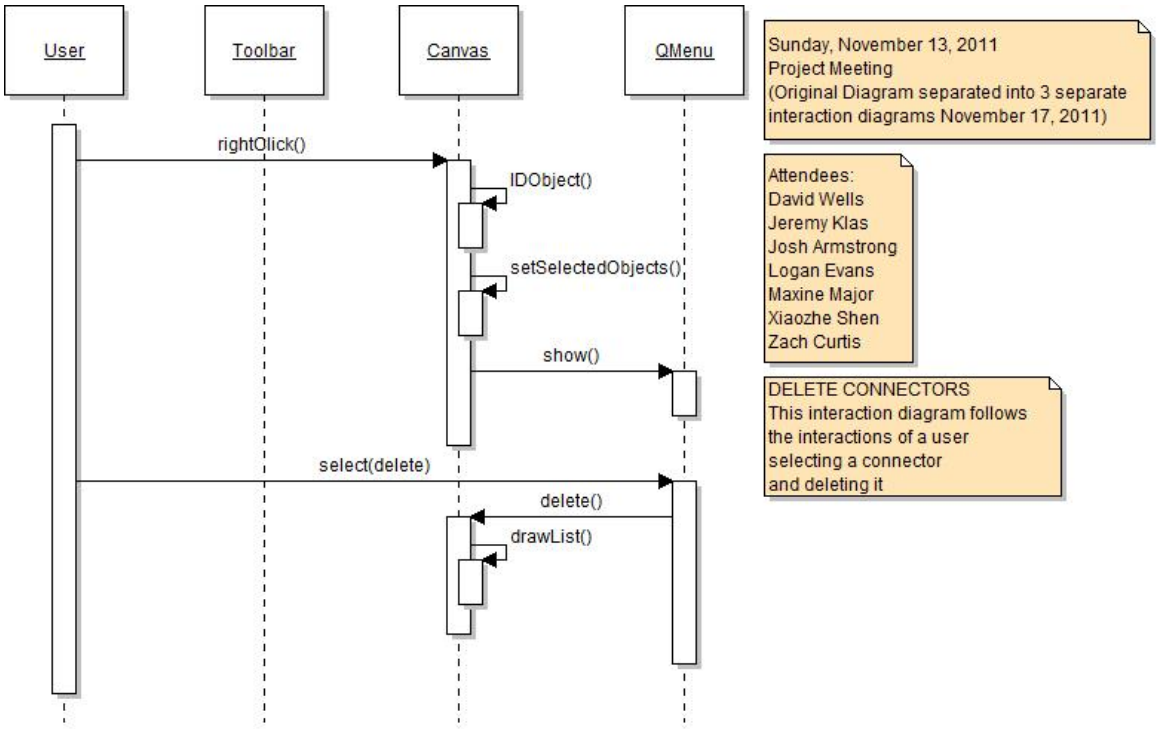


Figure 14: Delete Connector

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		testinstall	PASS		
Launch	2.2.1	pUML software launches successfully	M		testlaunch	PASS		
Exit	2.2.1	pUML Exit options	M		testsave * testsaveas *	FAIL FAIL		
Main Window	2.2.2	All main window features	M		testmainwindow	PASS		
Objects	2.2.3	Object properties and functionality	M		testobjects	PASS		
Connectors	2.2.4	Connector properties and functionality	M		testconnectors testselfconnector	FAIL PASS	PASS	
Open	2.2.5	Load a pUML file into pUML	M		testsave * testsaveas *	FAIL FAIL		
Save/Save As	2.2.6	Save a diagram	M		testsave testsaveas	FAIL PASS		
Diagrams	2.2.7	Diagram type integrity	M		testsave * testsaveas *	FAIL FAIL		
Tabs	2.2.8	Tab functionality	M		testsave * testsaveas *	FAIL FAIL		

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.