Bilkent University

CS 319
Object-Oriented Software Engineering

Fall 2009

Logic Design and Breadboard Simulator

Final Report
21 November 2009

Group #2

20602112	korpeoglu@ug.bilkent.edu.tr	Erdinç	Körpeoğlu
20702358	battal@ug.bilkent.edu.tr	Mustafa	Battal
20601393	m_zengin@ug.bilkent.edu.tr	Mustafa	Zengin
20601456	sarimurat@ug.bilkent.edu.tr	Salim	Sarımurat

Table of Contents

Table of Figures

1. INTRODUCTION

Object-oriented software engineering is a relatively new concept. It is a way of modeling the real structured systems into software based applications. Even though many builds up the idea that software engineers are just a bunch of coders, it is completely the other way. The design stage is a whole process by itself and it is probably the most intricate one in completing software project.

Our project is called Logic Design and Breadboard Simulator in long, but we also call it BBSim which is stands for Breadboard Simulator. As it is understood by the name, what system mainly does is to help user to make logic design of circuits and simulate them on a virtual breadboard.

In the rest of the report, problem statement, requirements analysis and analysis models are explained in detail. In the problem statement, problem that we tried to solve, goals of the project and works needed to be done are described. Moreover, in the requirements analysis, overview of the project, functional and non-functional requirements, constraints, scenarios, use case models and user interfaces are included. Lastly, object model and dynamic model of the system are mentioned in the analysis models part. Object model consists of domain lexicon and class diagrams. Apart from that, state charts and sequence diagrams are included in dynamic model of the system.

2. PROBLEM STATEMENT

Most of the Computer Science Departments of Colleges offer course(s) about Digital Circuit design and implementation. Assignments and Projects given in such courses involve designing, testing and creating digital systems where interfacing of input and output devices is occasionally necessary. However, using such kind of devices has some limitations as;

- The interfacing Workstations are only available in the department laboratories. So, it is not possible to work on the given assignment or project outside of the laboratories.
- The given tasks require a direct interaction with electronic hardware, such as Logic Gates, Wires, and Batteries. Infrequently, these hardware components break or run out. Under such circumstances, Students are not able to test their design to detect the mistakes before handling the given task.
- Such courses are generally mandatory in Computer Science Departments; so lots of Students are involved in them. While designing, testing and implementing the given task, Students working in laboratories have to be supervised by experienced Student or Staff. Because of this, the available experienced Student or Staff to help Students is limited. Therefore, there is a time limitation on the provided work sessions.

We have faced with these problems in CS 223 – Digital Design course and also thought of having a program that virtualizes the lab environment. Then, while taking CS 319 – Object Oriented Software Development course, we have decided to develop a project that would make it possible to design, build and test the given tasks outside of the laboratory. After we decided to work on project BBSim, we talked with our CS 223 Instructor (Dr. Özcan Öztürk) and asked his opinion about the project. Thanks to him, he directed us and suggested some other problems and features that we have not yet considered. He also said that it might be possible to use such a program in CS 223 course as well. Hearing this, we have fixed on working on the project BBSim and set our goal to develop an extensive program that will be worthwhile for Students who will take CS 223 course in the future.

We would like build a project that virtualizes lab environment and make it possible for Students to develop their Digital Design tasks effortlessly before building them on an actual Breadboard. The program should provide all of the functionalities of Digital Designing Concept such as Truth Table, Karnaugh Map, Logic Function Generation, Circuit Schematic Drawing and Breadboard Circuit Building.

Further, the program should furnish Verification of functionalities across each other and Simulation of the Built Breadboard Circuit. Since it is possible to do one of these functionalities, starting from any of these steps shall be available. At last, Student should be able to print any component of the project that he has developed.

3. REQUIREMENTS ANALYSIS

3.1. Overview

The aim of our project Logic Design and Breadboard Simulator is to provide a virtual environment to design and build logic circuits on breadboard. Designing and building a logic circuit include various processes. These processes could be described as circuit design, drawing circuit schematic, building circuit on breadboard, simulating circuit and verifying correctness and completeness of the circuit. We intended to include all of these processes in our project. In fact, all these steps are related to each other. To exemplify, the data which is created in circuit design may possibly be needed to draw circuit schematic. Therefore, program will provide interactions between different phases. Additionally, a new project must be created for each single circuit which user is intended to work on. Each project could be saved and loaded back. Apart from these issues, other features of the program are mentioned below.

Firstly, circuit design consists of three stages. These stages are drawing truth table, drawing karnaugh maps and generating logic functions from generated karnaugh maps. Our project is designed to handle these three steps to provide a complete circuit design environment.

After designing a circuit, the next step is to draw its schematic with gates and wires. According to our design, our program will enable users to draw circuit schematic for any circuit. Actually, circuit schematic is a step of whole system between designing circuit and building circuit.

Building circuit on breadboard includes three stages; selecting chips, locating them and making required connection through wires. By the help of our program, users will be able to apply these three steps on a virtual breadboard. On exceptional situations, program will provide removal of any component. Moreover, our program enables users to trace any part of the circuit. To exemplify, a user may want to observe existence of electric signal on a wire or on a breadboard socket, then s/he could check that component by trace feature of our program.

When a circuit is built on breadboard, outputs of the circuit are observed on the circuit simulation step. In that phase, user enters different input combinations and checks corresponding outputs. By that way, user observes whether Truth Table and output results are matched or not. Our program will provide that simulation mechanism.

As last step of whole system, we intended to add a verification feature to our program. The aim of that feature is to check correctness and completeness of a circuit which is designed and builded. In fact, user could make verification between truth table and circuit built on breadboard via simulation mechanism. However, we decided to provide other verification options on the program. These verification options are verification between karnaugh map and truth table, verification between generated logic functions and truth table, verification between circuit schematic and generated logic functions, also verification between circuits built on breadboard and circuit schematic.

In fact, the main part of the whole system is explained above. Together with that, our program will have get help and print features. By the get help feature, program will provide online and offline help to the user. If user does not have internet connection, s/he will be able benefit from static help content of the program. However, if user has internet connection, then s/he will be able to benefit from web sources.

Apart from the get help feature, program will have a print feature to enable users to print truth tables, karnaugh maps, circuit schematic, and actual circuit and simulation results.

In conclusion, there are many features of the whole system. Our aim is to develop a simulator which provides to implement all steps of designing and building a logic circuit. Apart from that, we are intended to develop a user friendly program that complexity of its contents will not cause a difficult usage.

3.2. Functional Requirements

Manage Project

Student should be able to create, save and load a project at any time as he desires. If Student has not entered the input and output specifications, Program should direct him before Student starts working on the project.

Draw Truth Table

Student should be able to fill the Truth Table which is generated by the Program automatically.

Draw Karnaugh Map

Student should be able to fill the Karnaugh Map(s) which is generated by the Program automatically.

Generate Logic Functions

Student should be able to generate Logic functions using the Karnaugh Map(s) that he has generated previously.

Draw Circuit Schematic

Student should be able to draw the Circuit Schematic using Logic representations of necessary Wire and Gate components.

Build Breadboard Circuit

Student should be able to build the circuit on a virtual Breadboard by using necessary Chip and Wire components.

Trace Circuit

Student should be able to trace the work that he has done so far on the breadboard to check whether he is moving correctly or not.

Simulate Circuit

Student should be able to simulate the circuit that he has built on Breadboard.

Verify

Student should be able to check correctness and coherence of the project that he has built.

Get Help

Student should be able to get help about the usage of the program, see the hotkeys and their functionality in the program, and the features of the components.

Print

Student should be able to print any part of the work that he has done. He should be able to set the properties of the page to be printed as well.

3.3. Non-functional Requirements

3.3.1. Usability

Student should be directed to follow the subsequent successive steps while designing and building his circuit.

- Enter the number of inputs and outputs together with their names.
- Draw Truth Table.
- Draw Karnaugh Map.
- Generate Logic Functions.
- Design Circuit.
- Build Circuit on Breadboard.
- Simulate Circuit.
- Verify correctness of system.

However, following this path should not be obligatory. Student should be able to start designing and building his project at any independent step. For example, it is possible to start designing the circuit before drawing the truth table. However, if he chooses to design in this way, he will not be allowed to verify his circuit using Truth Table because they are dependent on each other.

3.3.2. Reliability

If System is interrupted at any step because of a crash, Student should be able to load the project without any data loss later.

3.3.3. Implementation

The program should run on both Linux and Windows machines. This suggests that the program should ensure ease of portability between different environments.

3.4. Constraints

1. Program should be implemented using Java. Because; it is a highly convenient programming language for Object Oriented approach, its performance is satisfactory, and it has many different supporting libraries that will shorten the time required for the project delivery.
2. Program should not have any functionality that is done automatically by the system. Since the goal of this project is to educate students to design and build logic circuits, automated processes are prevented. The aim is to direct students to learn the whole concept. To exemplify, after student draw truth table, Karnaugh map values could be filled by system automatically, however, it is not allowed in order to teach students how to fill Karnaugh maps.

3.5. Scenarios

3.5.1. Scenario Name: Tail Light Project Creation

Participating Actors:

Ahmet: Student

Flow of Events:

1. Ahmet has a laboratory assignment of designing a digital circuit for car tail lights. He was expected to draw Truth Table, Karnaugh Map(s), and Circuit Design and to build On-Board Circuit. Additionally, he is expected to simulate his circuit. He opens the program and activates "Create Project" functionality.

2. He specifies the project name as Taillight and the directory as his older workspace folder which was defined at the first usage of BBSim program.
3. He specifies input and output numbers and their names. Given assignment have three inputs and three outputs. Input names are "Brake", "Rear", "Signal" and output names are "RedLed", "WhiteLed", "YellowLed".
4. Ahmet waits the system to create and load the project.

3.5.2. Scenario Name: Review Project

Participating Actors:

Ahmet: Student

Flow of Events:

1. Ahmet was working on his circuit, and then he got tired and wanted to go to sleep. Therefore, he decided to save and close his project.
2. To save and close the project, he pushes “Close” button of the program BBSim and then program asks him whether he wants to save changes or not.
3. Ahmet pushes “Save” button, and then project is saved and program is closed.
4. Next morning, Ahmet wanted to work on his project again.
5. He opens the program and he invokes “Load Project” functionality to load his project directory.
6. He selects his project directory from the workspace that holds projects that Ahmet has created.
7. At the end, Ahmet has his previously saved project ready to be worked on.

3.5.3. Scenario Name: Drawing Truth Table for Tail Light Project

Participating Actors:

Ahmet: Student

Flow of Events:

1. After the project initiation process is completed, Ahmet wants to design the circuit that he is assigned. And, he invokes the “Truth Table Drawing” functionality.
2. As the first step of the whole project, Ahmet has already informed the System that there are 3 inputs and 4 outputs in the circuit. By that information, System generates a Truth Table which includes 3 input columns and 4 output columns. Additionally, System automatically fills up the input columns to make Ahmet's job easier.
3. Ahmet fills output columns according to his design solution.
4. At the end, Ahmet pushes “Done” button to complete “Truth Table Drawing” process.

3.5.4. Scenario Name: Drawing Karnaugh Map(s) for Tail Light Project

Participating Actors:

Ahmet: Student

Flow of Events:

1. Ahmet wants to draw Karnaugh Maps for his circuit, therefore he activates “K-Map drawing” functionality.
2. As the first step of the whole project, Ahmet has already informed the System that there are 3 inputs and 4 outputs in the circuit. By that information, System generates 4 different Karnaugh Maps each of which is constructed for one single output.
3. Names of K-Maps are held in a list. Ahmet selects the name of a K-Map from the list as he desires. Then, selected K-Map becomes visible on the screen.
4. Ahmet fills K-Map in the guidance of the Truth Table.
5. For each K-Map Ahmet repeats 3^{rd} and 4^{th} steps to complete K-Map drawing process.
6. At the end, Ahmet pushes “Done” button to finish the whole process of K -Map drawing.

3.5.5. Scenario Name: Generating Logic Functions of Tail Light Project from K-Maps

Participating Actors:

Ahmet: Student

Flow of Events:

1. Ahmet activates “Generate Logic Function from K-Map” functionality.
2. In that step, K-Maps which are generated in “K-Map Drawing” process are used to derive logic functions that represent each single output of Tail Light circuit. Ahmet's circuit has 4 outputs which are named as T, X. Y and Z. Therefore, Ahmet needs to generate 4 different logic functions to represent these 4 outputs.
3. Names of 4 outputs are held in a list. Ahmet selects output X from the list. Then, previously generated K-Map of output X becomes visible on the screen. Additionally 2 boxes appear on the screen. One of the boxes holds unoptimized logic function of output X. The other box holds optimized logic function of output X.
4. Ahmet groups the cells which hold 1 (as binary value, meanly logical “true”) on the K-Map.
5. To generate Sum Of Products (SOP) form of a group, Ahmet pushes “Transfer” button to transfer logical representation of that group into the box that is assigned to hold unoptimized logic function of output X.

6. Ahmet continues to group cells and transfer their logical representations to unoptimized formula box until the last cell which holds 1 (as binary value, meanly logical “true”) is included in a group.
7. Ahmet calculates optimized logical expression for the output X manually.
8. Ahmet enters optimized logical expression to the box that is assigned to hold optimized logic function of output X.
9. Ahmet generates unoptimized and optimized logical expressions for output T, Y and Z by repeating steps from 3 to 8, in the same manner.
10. Ahmet pushes “Done” button and a message appears on the screen saying “Logic Function generation process is completed.”
11. Ahmet pushes “OK” button on the message and he completes all the process.

3.5.6. Scenario Name: Draw Circuit Design

Participating Actors:

Ahmet: Student

Flow of Events:

1. Ahmet activates “Drawing Circuit Design” functionality.
2. In the guidance of logic functions that are generated manually by Ahmet or automatically by “Generate Logic Function from K-Map” functionality, Ahmet starts drawing the design of the circuit using the distinctive shapes of the logics gates.
3. Ahmet uses AND, OR, NAND, XOR, NOR, NOT Gates and Wire as design components to design circuit.
4. At first, Ahmet selects an AND gate as a design component and locates that AND gate by dragging and dropping it on the design screen as he desires. In the same manner, Ahmet uses the other required Gates.
5. Ahmet selects Wire component and connects the Gates on the circuit.
6. To complete the “Drawing Circuit Design” session, Ahmet needs to assign a Chip Type and the Pin Numbers for all of the Gates in the circuit. Successively, Ahmet clicks on the Gates and enters a Chip Type and pin numbers for the inputs and output of every Gate.
7. Ahmet pushes “Done” button and a message appears on the screen saying “Drawing Circuit Design process is completed.”
8. Ahmet pushes “OK” button on the message and he completes all the process.

3.5.7. Scenario Name: Build on board circuit of taillight project

Participating Actors:

Ahmet: Student

Flow of Events:

1. After Ahmet has completed design of circuit, he wants to build that circuit on the bread board. Therefore, he activates “Build Circuit on BB” functionality.
2. At first, Ahmet wants to make Power and Ground connections of the bread board. To make Power connection, he selects wire component to be added into circuit. Then, he locates one end of the wire to the Power socket of the workstation and locates other end of wire to the electric (+) line of the bread board. Additionally, to make Ground connection, Ahmet selects wire component to be added into circuit again. Then, he locates one end of the wire to the Ground socket of the workstation and locates the other end of wire to the Ground line of the bread board. To locate a wire, after wire is selected to be added into circuit, Ahmet determines its start point by clicking a specific point and then he determines end point by stretching the wire until desired end point is reached.
3. Ahmet selects 74LS08 Quad 2-Input AND Gate chip from chip list. Then he locates that chip on the breadboard.
4. Ahmet makes power and ground connections of the chip. To achieve power connection, Ahmet connects power pin of the chip to electric (+) line of the breadboard through a wire. Additionally, to achieve ground connection, Ahmet connects ground pin of the chip to ground (-) line of the breadboard through a wire.
5. After power and ground connections of the chip are done, Ahmet wants to make input connections of the circuit. To make a single input connection, he selects wire component to be added into circuit. He locates one end of wire to the one of the input sockets of workstation and locates other end of wire to the required pin of the chip. In fact, other input connections are made by the same way.
6. As next step, Ahmet wants to make internal connections of the circuit. Internal connections provide interaction between different gates and chips. To achieve a single connection, he selects wire component to be added into circuit. Then, he locates two ends of the wire on required pins of the chip. Other internal connections of the circuit are made by the same way.

7. Lastly, Ahmet makes output connections of the circuit. To achieve an output connection, Ahmet selects wire component to be added into circuit. Then, he locates one end of the wire to a pin of the chip which gives output of the circuit, and locates other end of wire to a Led which indicates situation of that specific output. Other output connections could be made by the same way.

3.5.8. Scenario Name: Simulate BB Circuit of Tail Light for Brake

Participating Actors:

Ahmet: Student

Tayfun: Assistant

Flow of Events:

1. Once Ahmet completes his circuit generation on the Breadboard, he wants to test the project outputs for brake operation. He pushes the “ON/OFF Simulation” button on the screen when it is in “OFF” state.
2. Ahmet checks the list generated by the program which stores the expected outputs that he has entered in the Truth Table Drawing functionality previously and the actual outputs of the circuit that he has built on the breadboard.
3. He compares the output values on the table and observes the differences between them.
4. After checking the list, Ahmet wants to visualize outputs of different input combinations on the output Leds. He assigns the input values by using switches on the workstation. These input values correspond to the Break operation.
5. As Ahmet assumes, all output Leds turn on.
6. After that, Ahmet wants to simulate the circuit for input values which does not correspond to the brake operation.
7. While he is changing the input switches, the output Leds are automatically turns ON or OFF. By changing the input values, Ahmet checks whether there are any unexpected behavior on the circuit or not.
8. He realizes that 2 outputs are not working as expected. As it is determined on the list.
9. Ahmet pushes the “ON/OFF Simulation” button on the screen when it is in “ON” state and continues to build his circuit on breadboard to fix the problems that he has realized.

3.5.9. Scenario Name: Verify Circuit Design by Logic Functions

Participating Actors:

Ahmet: Student

Tayfun: Assistant

Flow of Events:

1. After Ahmet has completed design of the circuit, he wants to show his design to the Assistant Tayfun.
2. Tayfun activates the "Verification" functionality and he selects to verify Ahmet's circuit design by matching it with the Logic Functions that Ahmet has generated by using "Generate Logic Function from K-Map" functionality.
3. System generates a report on the screen. This report lists the parts of the function that are not included in the design.
4. Tayfun wants Ahmet to check his design once more and correct his mistakes.

3.5.10. Scenario Name: Trace Circuit of Tail Light for Brake

Participating Actors:

Ahmet: Student

Tayfun: Assistant

Flow of Events:

1. While Ahmet continues to build his circuit on Breadboard, he wants to trace his circuit step by step. Therefore, he decides to check the existence of electricity on wires and breadboard sockets. The system enables users to follow signals (i.e. 1s and 0s) in any point of the circuit built on the breadboard He pushes the "ON/OFF Tracing" button on the screen when it is in "OFF" state.
2. Ahmet assigns the input values by using switches on the workstation.
3. After input values are assigned, Ahmet clicks on a wire which he wants to learn whether signal is available on that wire or not.
4. When wire is clicked, Ahmet observes that Trace Led turns on. It means that signal is available on that wire.
5. Additionally, Ahmet wants to check signal on a chip leg, therefore, he clicks on the corresponding socket of the chip leg.
6. Ahmet observes an expected absence of signal on the chip leg because Trace Led turns off.

7. Since Ahmet observes the signals as expected, he decides to continue building the other parts of the circuit.
8. He pushes the “ON/OFF Tracing” button on the screen when it is in “ON” state and continues to build his circuit on breadboard.

3.5.11. Scenario Name: Get Help

Participating Actor:

Ahmet: Student

Flow of Events

1. Ahmet is a student who recently started to use BBsim. Therefore, he needs some help about the usage of the program. For example, building circuit on bread board could be a confusing issue about selecting components and locating them on the circuit.
2. Ahmet accesses one of the following options of getting help and gathers information.

- Read Offline Manual
- Get Online help about BBSim

3.5.12. Scenario Name: Print circuit design of Taillight project

Participating Actor:

Ahmet: Student

Flow of Events

1. Ahmet wants to have a hard copy of his circuit design for Tail Light project so that he can add it to his preliminary work of assignment. He gets "print document" property of system started.
2. While system generates printer friendly format of the circuit design file, Ahmet fills the form about details of printing operation - high quality printing, one copy and 1-to-1 scaling on A4 paper. He confirms that information.
3. Ahmet reviews final version before printing, he finds 1-to-1 scaling a little bit small for an A4 paper, and he goes back and changes it to 1-to-2. He confirms and reviews again.
4. He orders system to send file to printer, after circuit design is printed; he staples it to the rest of his work.

3.6. Use Case Models

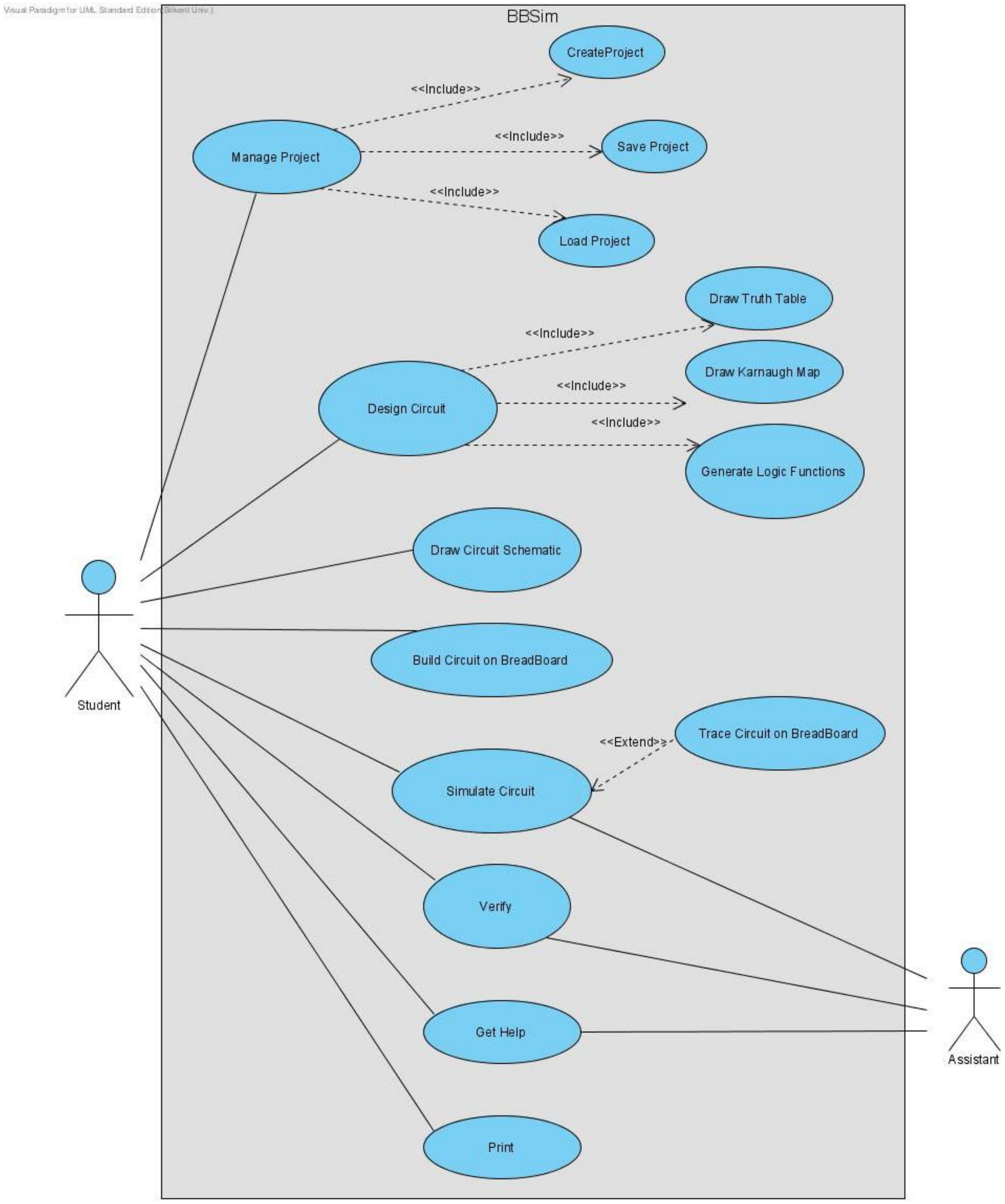


Figure 1 - Use Case Drawing

3.6.1. Name of Use Case : Manage Project

Participating Actors : Initiated by Student

Entry Condition:

- Student initiates BBSim program.

Flow of Events:

1. Student creates a new project. (Extended use case “Create Project”)
2. Student saves his project. (Extended use case “Save Project”)
3. Student loads his project. (Extended use case “Load Project”)

Exit Condition:

- Student continues with designing the circuit.
- Student closes the program BBSim.

* * *

3.6.2. Name of Use Case : Create Project

Participating Actors : Initiated by Student

Entry Condition:

- Student creates a workspace to store his BBSim projects.

Flow of Events:

1. Student activates "Create Project" functionality.

2. Student specifies the project name and the project directory inside his workspace folder.

3. Program asks for the number of inputs and outputs of the circuit that is going to be designed.

4. Student enters the number of inputs and outputs.

5. Program asks for the name of the inputs and outputs.

6. Student enters names for the inputs and outputs.

7. Program generates the project file with the given entries for inputs and outputs.

Exit Condition:

- Student cancels the project creation process.
- Student confirms the project creation process.

3.6.3. Name of Use Case **:** Save Project

Participating Actors **:** Initiated by Student

Entry Condition:

- Student hits the “Close Window” button before saving his design.

Flow of Events:

1. Student activates "Save Project" functionality.

2. Program asks Student whether he wants to save changes or not.

3. Student selects to “Save” the project.

4. Program saves the project.

Exit Condition:

- Program is closed.

Exceptions:

1. Student pushes “Don't Save” button to reject saving the project and close the program.

2. Program does not save the project.

3. Student selects to “Cancel” and reject saving the project and Continue working on the program.

* * *

3.6.4. Name of Use Case **:** Load Project

Participating Actors **:** Initiated by Student

Entry Condition:

- Student hits “Load” button.

Flow of Events:

1. Student activates "Load Project" functionality.

2. Program asks Student to specify the project file and directory to be loaded.

3. Student selects the project file and directory.

4. Program loads the specified project.

Exit Condition:

- Student has his previously saved project ready to be worked on.

3.6.5. Name of Use Case **:** Design Circuit

Participating Actors **:** Initiated by Student

Entry Condition:

- This use case starts right after the project is created.
- This use case starts right after the project is loaded.

Flow of Events:

1. Student draws the Truth Table for his circuit. (Extended use case “Draw Truth Table”)

2. Student draws the Karnaugh Map for his circuit. (Extended use case “Draw Karnaugh Map”)

3. Student generates the Logic Functions for his circuit. (Extended use case “Generate Logic Functions”)

Exit Condition:

- Student continues with drawing the circuit schematic.
- Student closes the program BBSim.

* * *

3.6.6. Name of Use Case **:** Draw Truth Table

Participating Actors **:** Initiated by Student

Entry Condition:

- Student hits the “Draw Truth Table” button on the screen.

Flow of Events:

1. Student activates "Truth Table Drawing" functionality.

> **2.** Program loads the “Truth Table Drawing” functionality using inputs and outputs which are specified in project creation phase. A truth table with the given inputs and outputs are drawn on the screen.
>
> **3.** Program automatically fills the input columns of the table.

4. Student fills the output columns of the table.

5. Student selects “Done” to complete “Truth Table Drawing” process.

> **6.** Program prints a message on the screen saying “Truth Table Drawing process is completed.”

Exit Condition:

- Student pushes “OK” button on the message and finishes the process.

Exceptions:

1. After “Done” is selected, if output columns are not filled fully, then a message appears on the screen saying “There are unfilled cells on the Truth Table, go and fill them.”

2. Student turns back to Truth Table drawing activity and fills the empty output cells.

3. Student pushes “Done” button again and finishes the process.

*   *   *

3.6.7. Name of Use Case **:** Draw Karnaugh Map(s)

Participating Actors **:** Initiated by Student

Entry Condition:

- Student hits the “Draw Karnaugh Map(s)” button on the screen.

Flow of Events:

1. Student activates "Drawing Karnaugh Map(s)" functionality.

2. Program loads the “Karnaugh Map Drawing” functionality using inputs and outputs which are specified in project creation phase. For every single output, a Karnaugh Map is generated and drawn on the screen.

3. Program stores the names of the K-Maps in a list which is visible to the user.

4. Student selects the name of a K-Map from the list as he desires. Then, selected K-Map becomes visible on the screen.

5. Student fills K-Map in the guidance of the Truth Table.

6. For each K-Map, Student repeats 3rd and 4th steps.

7. Student selects “Done” to complete “Karnaugh Map Drawing” process.

8. Program prints a message on the screen saying “Karnaugh Map Drawing process is completed.”

Exit Condition:

- Student pushes “OK” button on the message and finishes the process.

Exceptions:

1. After “Done” is selected, if all of the K-Maps are not filled entirely, then a message appears on the screen saying “There are unfilled cells, go and fill them.”

2. Student turns back to K-Map drawing activity and fills the empty cells.

3. Student selects “Done” again and finishes the process.

3.6.8. Name of Use Case : Generate Logic Function(s)

Participating Actors : Initiated by Student

Entry Condition:

- Student selects to start "Generate Logic Function(s)".

Flow of Events:

1. Program loads "Generate Logic Function(s) from K-Map" functionality using the Karnaugh Maps that are generated in "K-Map Drawing" process.

2. Program stores the names of the outputs in a list which is visible to the user.

3. Student selects the name of an output from the list.

4. Program loads corresponding Karnaugh Map of the selected output and 2 formula boxes on the screen. These boxes are assigned to hold Unoptimized and Optimized Logic Functions of the selected output in textual format.

5. Student groups the cells which hold 1 (as binary value, meanly logical "true") on the K-Map.

6. To generate Sum of Products (SOP) form of a group, Student pushes "Transfer" button to transfer logical representation of that group into the box that is assigned to hold Unoptimized Logic Function of the selected output. In fact, logical representation is generated by the program.

7. Program gives different colors to different cell groups which their SOP terms are generated.

8. Student continues to group cells and transfer their logical representations to Unoptimized formula box until the last cell which holds 1 (as binary value, meanly logical "true") is included in a group.

9. Student calculates optimized logical expression for the output X manually.

10. Student enters optimized logical expression to the box that is assigned to hold optimized logic function of output X.

11. Student generates Unoptimized and Optimized logical expressions for every output by repeating steps from 3 to 8, in the same manner.

12. Student selects "Done" to complete "Logic Function Generation" process.

13. Program prints a message on the screen saying "Logic Function Generation process is completed."

Exit Condition:

- Student pushes “OK” button on the message and finishes the process.

Exceptions:

A.1. After “Done” button is pushed, if all of the K-Maps are not filled entirely, then a message appears on the screen saying “There are unfilled cells on the K-Map, go and fill them.”

A.2. Student turns back to K-Map drawing activity and fills the empty cells.

A.3. Student pushes “Done” button again and finishes the process.

B.1. If Student tries to activate “Generate Logic Function(s) from K-Map” functionality before he completes the “K-Map Drawing” phase, he is warned with a message saying “Please proceed through K-Map drawing functionality.”

C.1. After “Done” button is pushed, if Student has left any 1 (as binary value, meanly logical “true”) that is not included in any group of SOPs, and then a message appears on the screen saying “Please go and check the K-Map(s). There are some unmatched 1's in the K-Map(s).”

C.2. Student returns back to “Generate Logic Function(s) from K-Map” functionality and become sure that all 1's are included in a group.

C.3. Student pushes “Done” button and finishes the process.

* * *

3.6.9. Name of Use Case **:** Draw Circuit Schematic

Participating Actors **:** Initiated by Student

Entry Condition

- Student selects to enter “Draw Circuit Schematic”.

Flow of Events:

1. Program loads “Drawing Circuit Schematic” functionality.

2. Program displays the Logic Functions that are generated manually by Student or automatically by “Generate Logic Function from K-Map” functionality on the screen.

3. Student selects and locates distinctive shapes of Logic Gates which are AND, OR, NAND, XOR, NOR, NOT as design components.

4. Student selects Wire component and connects the Gates on the circuit.

5. Student assigns a Chip Type and the Pin Numbers for each Gate in the circuit schematic. Successively, Student clicks on the Gates and enters a Chip Type and pin numbers for the inputs and output of every Gate.

6. Student "Done" is selected to complete "Drawing Circuit Schematic" process.

7. Program prints a message on the screen saying "Drawing Circuit Schematic process is completed."

Exit Condition

- Student selects "OK" on the message and he completes all the process.

Exceptions:

A.1. Student has left some disconnected inputs or outputs on a Gate.

A.2. Program warns the user with a message saying "Please review the circuit schematic."

A.3. Student returns back to the "Drawing Circuit Schematic" process.

B.1. Student realizes a component on the circuit schematic that is not necessary.

B.2. Student clicks on that component and then pushes "Remove Component" button.

B.3. Program removes that component from the screen.

* * *

3.6.10. Name of Use Case **:** Build Circuit on Breadboard

Participating Actors **:** Initiated by Student

Entry Condition

- Student selects the "Build Circuit on Breadboard".

Flow of Events:

1. Program loads the "Building Circuit on Breadboard" functionality.

2. Student makes Power and Ground connections from Workstation to the Bread Board using Wire components.

3. Student selects Chips from Chip List and locates them on the Breadboard.

4. Student makes Power and Ground connections of all Chips by using Wire components.

5. Student makes input connections of the circuit from the Workstation to the Chips on the Breadboard by using Wire components.

6. Student connects the Gates to each other internally by using Wire components.

7. Student connects output(s) of the circuit to the Led(s) on the Workstation by using Wire components.

Exit Condition:

- Student finishes building his circuit and decides on "Simulate Circuit".

Exceptions:

A.1. Student realizes a component on the Breadboard that is not necessary.

A.2. Student clicks on that component and then decides on "Remove Component".

A.3. Program removes that component from the screen.

B.1. Student double clicks on any Chip.

B.2. Program shows the pin assignments of that Chip on a picture.

*   *   *

3.6.11. Name of Use Case **:** Simulate Circuit

Participating Actors **:** Initiated by Student

Entry Condition:

- Student selects "ON/OFF Simulation" when it is in "Off" state.

Flow of Events:

1. Program loads the "Simulation" functionality.

2. Program builds a table of 3 columns based on the values coming from the "Truth Table Drawing" functionality and the circuit built on the breadboard. First column stores the input combinations, second column stores the outputs of these inputs coming from the Truth Table Drawing and the last column stores the output values of the circuit built on the breadboard for these inputs.

3. Student selects the input values (0 -false- or 1-true-) by using the switches on the workstation.

4. While input values are changing, the output Leds are automatically turns ON or OFF based on the circuit built on the breadboard.

Exit Condition:

- Student selects "ON/OFF Simulation" when it is in "On" state.

3.6.12. Name of Use Case **:** Verify

Participating Actors **:** Initiated by Assistant
Communicates with Student

Entry Condition:

- Student selects “Verify” on the screen.

Flow of Events:

1. Program loads the “Verification” functionality.

2. Student shows his circuit to the Assistant.

3. Assistant selects one of the following verification activities;

- Verify Karnaugh Map by Truth Table
- Verify Unoptimized Function by Truth Table
- Verify Circuit Schematic by Unoptimized Function
- Verify Circuit Schematic by Optimized Function
- Verify the Circuit on Breadboard by Circuit Schematic
- Verify the Circuit on Breadboard by Unoptimized Function
- Verify the Circuit on Breadboard by Optimized Function

and pushes the “Check” button.

4. Program checks whether the selected entities match or not.

5. Program generates a message on the screen saying “Success. Two entities match.” OR “Fail. Two entities do not match”.

Exit Condition:

- Assistant selects “OK” of the message screen. Then he can continue with verifying other steps or passing another phase of the system.

Extensions:

A.1. Student has generated the Logic Functions elsewhere, and they are not stored in the system.

A.2. Assistant wants Student to enter his logic functions to the system manually.

A.3. After Student has entered the logic functions to the system, Assistant proceeds through the “Verification” functionality starting from 2nd step to 4th step.

3.6.13. Name of Use Case **:** Trace Circuit

Participating Actors **:** Initiated by Student

Entry Condition:

- Student selects “ON/ OFF Tracing” on the screen when it is in “Off” state.

Flow of Events:

1. Program loads the “Trace Circuit” functionality.

2. Student assigns input values (0 -false- or 1-true-) by using the switches on the workstation.

3. Program turns ON or OFF the output Leds automatically based on the circuit built on the breadboard.

4. Student clicks on the wires or sockets on the breadboard.

5. Program turns the Trace Led ON or OFF on the workstation according to the existence of the electricity on the point clicked by the Student.

Exit Condition:

- Student pushes the “On/Off Tracing” button on the screen when it is in “On” state.

* * *

3.6.14. Name of Use Case **:** Get Help

Participating Actors **:** Initiated by Student
Initiated by Assistant

Entry Condition:

- Student selects “Help” on the screen.

Flow of Events:

1. Program opens a pop-up screen that stores the help information.

2. Student selects from Online or Off-line help options.

3. If Off-line help is selected, Program displays local help on the screen. Otherwise, Program opens the help documents that are stored on BBSim project website.

Exit Condition:

- Student hits the close button of the Pop Up screen.

3.6.15. Name of Use Case : Print

Participating Actors : Initiated by Student

Initiated by Assistant

Entry Condition:

- Student selects “Print” on the screen.

Flow of Events:

1. Student selects the page that he would like to print from one of the following options;

  - Truth Table
  - Karnaugh Map
  - Logic Functions
  - Circuit Design
  - Breadboard Circuit

2. Program opens a pop-up screen and asks the user about page layout, zooming amount, etc.

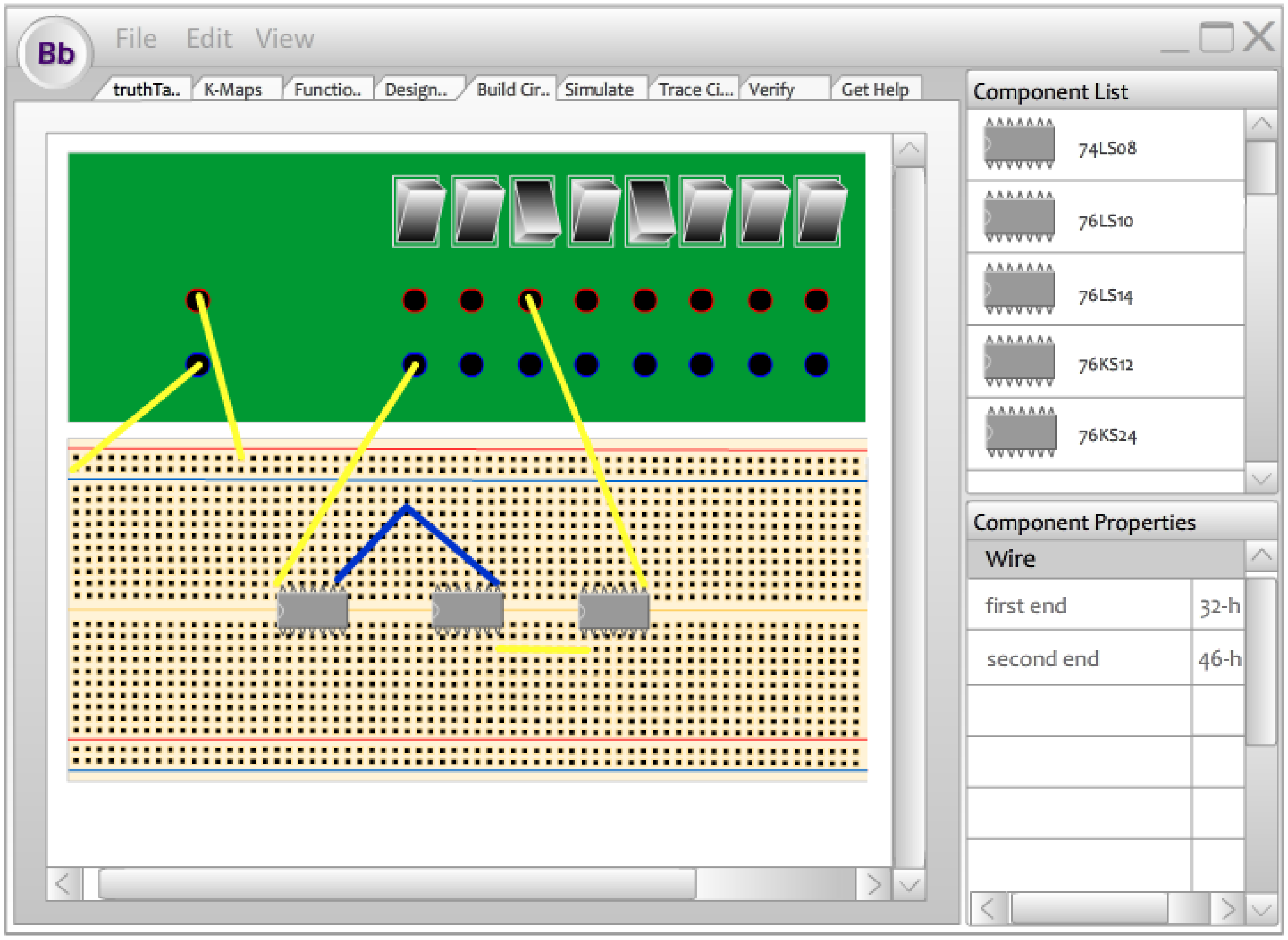
Exit Condition:

- The requested page is successfully printed.

3.6. User Interface Prototypes

The following prototypes are drawn to illustrate the User Interface of our project. It does not mean that we will organize the look and feel of our program exactly as these prototypes though.

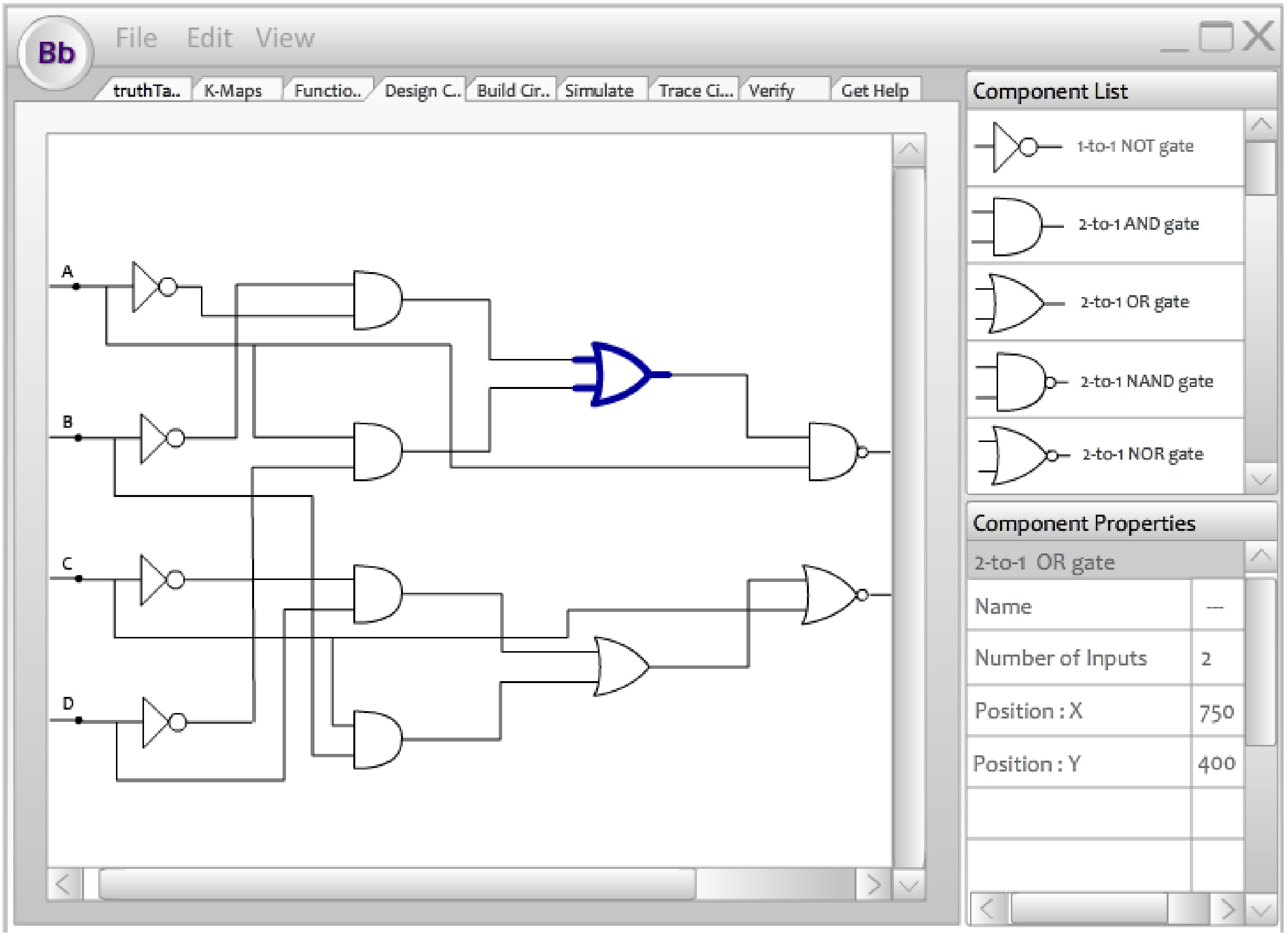
In Figure 2, Building Circuit Schematic on Breadboard step of our project is seen. On the main pane of this prototype, a workstation and a breadboard are seen. On the right pane, the chip set and the properties of the selected component is seen.



first end	32-h
second end	46-h

Figure 2 - User Interface Prototype for "Build Circuit Schematic on Breadboard"

In Figure 3, Designing Circuit Schematic step of our project is seen. On the main pane of this prototype, a design of the current user is seen. On the right pane, the chip set and the properties of the selected component is seen.

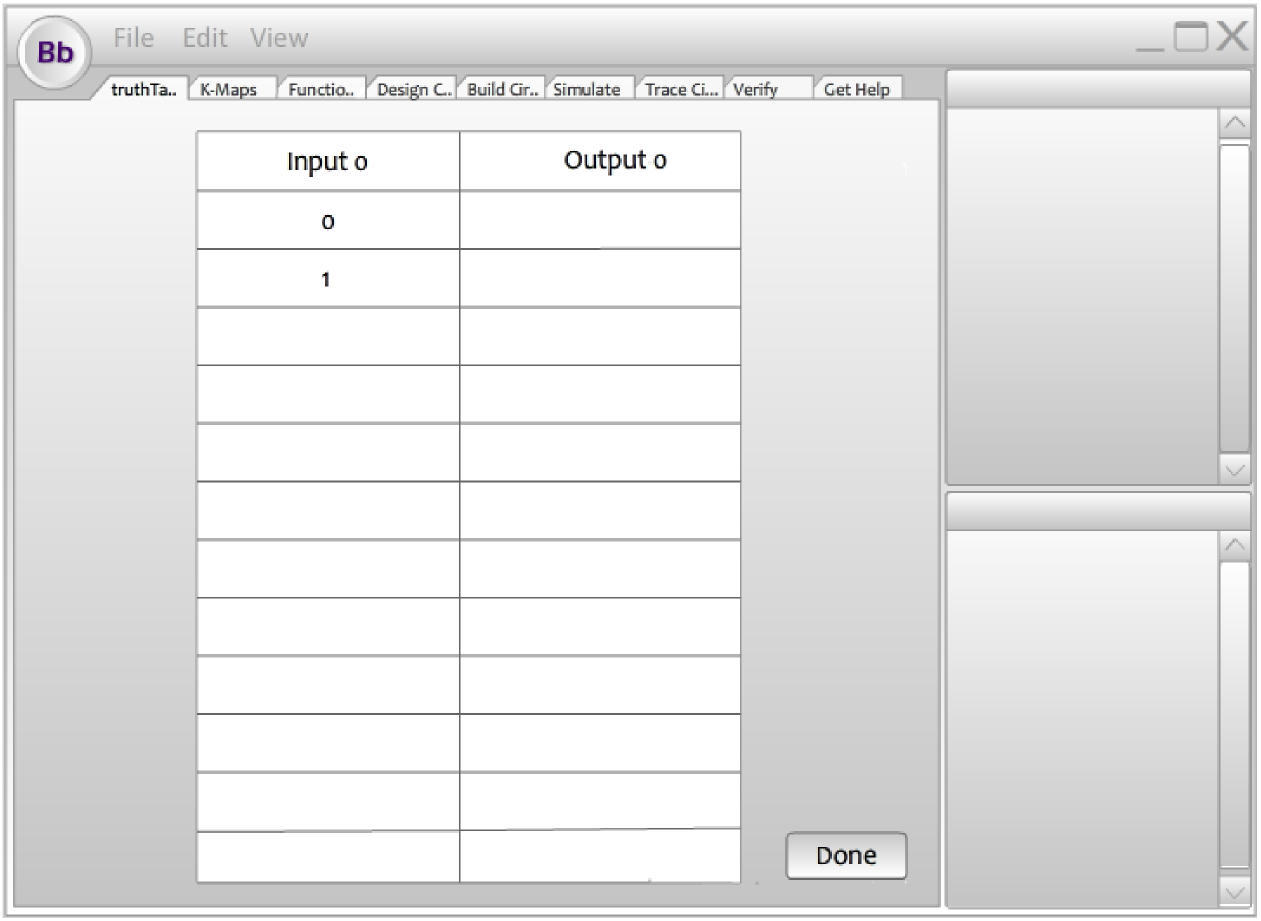


2-to-1 OR gate

Name	—
Number of Inputs	2
Position : X	750
Position : Y	400

Figure 3 - User Interface Prototype for "Design Circuit Schematic"

In Figure 4, Truth Table Filling step of our project is seen. On the main pane of this prototype, a truth table with the give inputs and outputs is seen.



Input o	Output o
0	
1	

Figure 4 - User Interface Prototype for "Truth Table Fill Out"

User clicks on IO Determine form (Figure -5) and enter the name and type (Input or Output) of the IOs.

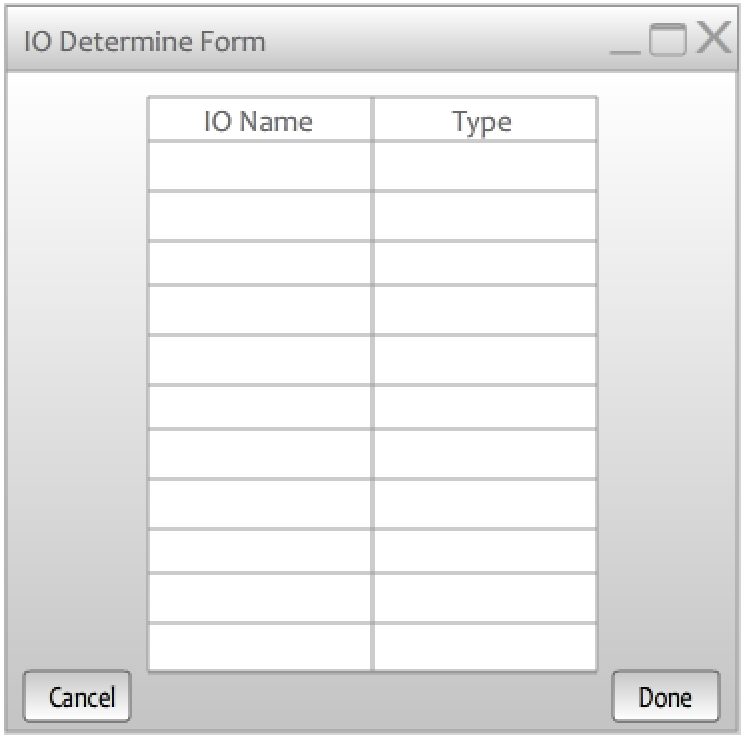


Figure 5 - User Interface Prototype for "IO Determine Form”

When the user is doing something that will affect some other parts of the project that he is building, we will warn him before processing the thing that he would like to do. If “Yes” button is clicked, program will process the thing that is expected by the user.

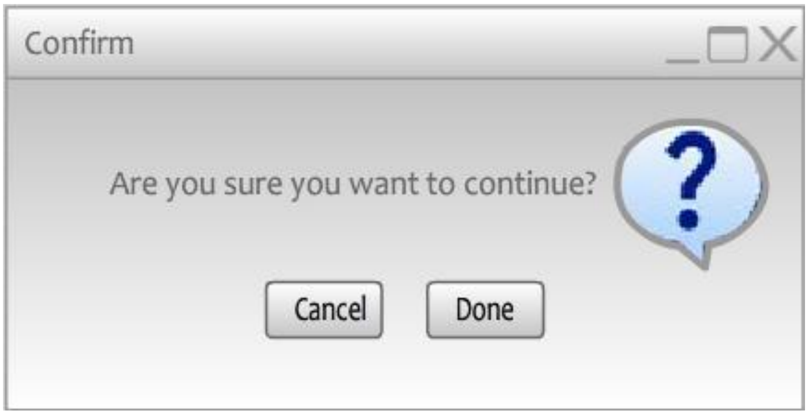


Figure 6 - User Interface Prototype "Confirm Message Pop Up Window

4. ANALYSIS MODELS

4.1. Object Model

We have explained the domain lexicons for each use case in different headings. After that, we have drawn their class diagram using the object that we have come up.

4.1.1. Domain Lexicon

Project

Project is simply a compilation of other sub-parts. It contains *TruthTable, KMap(s), Optimized & UnoptimizedLogicFunction, Circuit, Breadboard* and *Workstation.* Additionally, it has properties like name, time of creation like an actual file. It has a status (open, closed etc.) and it holds the input/output information that is used almost every step.

TruthTable

A table object that consists all input combinations of a circuit and corresponding output values for each input combination. Its input combination data is automatically generated an output data is filled by *Student*. It is possible to edit its data anytime during its lifetime.

KMap

A map object that includes boolean values of a specific output. It is used to simplify logic function expressions.

OptimizedLogicFunction

The simplest expression for a logic function. There could be one or more of such expressions.

UnoptimizedLogicFunction

Logic function expression derived from a *KMap*. Depending on the number of outputs in the *Project*, there can be one or more *UnoptimizedLogicFunction*s.

CircuitSchematic

It is the object that contains and holds all of the components together (*Wire*s, *Gate*s, *and IOs*). It can be either included in the *Project* or not.

IO

An *IO* represents predefined wire endings that are generated by input and output information provided by *User* in earlier steps. It has a name and a type (Input or Output). *IO* has snapping to *Wire* ability.

Wire

Simply corresponds to a wire. It has two ending points and variable number of breaking points. It can be bent over multiple times by breaking points, be resized, or be connected to different connection points during its lifetime.

Gate

Logic gates in real circuit. It has type (and, or, not etc.), chip type, pin number. It gets added to circuit, located in it, rotated around itself and removed from circuit. A *Gate* can make snapping connection with a *Wire*.

Workstation

This represents the I/O device used in laboratories. It provides *WorkstationSocket*s for input signals and output *Led*'s. Number of inputs and outputs on *Workstation* is determined in *IODetermineForm*. It has a switch for turning on and off which is used in *Simulation* use case.

Led

Led is connected to an output of the circuit to visualize the existence of signal at that output by turning ON or OFF. It also is a snap point for *Wire*s and has functionality of turning on and off according to electricity. It's colour is to be chosen from *ComponentPropertiesDisplay* area.

BreadBoard

It corresponds to real life BreadBoard. It contains many *BreadBoardSockets* and is connected to other components through these sockets. BreadBoard is free to move inside *BBEditPanel*. It enables *BreadBoardSockets* to build interconnection as they form regions.

BreadBoardSocket

Simply corresponds to any of the sockets on the *BreadBoard* object.

Chip

Real life chips with one or more logic gates inside. All type of Chips can be added, moved around, removed. It also can built connection. A *Chip* has type, logic gates and a number of pins.

SimulationEngine

SimulationEngine enables *Student* to simulate the circuit that he has built on BreadBoard. This engine makes it possible to see the *TruthTable* outputs and the circuit outputs at the same time. In this way, *Student* can observe any unexpected behavior of the circuit that he has built.

BBOutputTable

Table that stores the outputs of the associated *TruthTable* with current project. It is used to compare results tested in simulation with expected results.

VerificationEngine

VerificationEngine enables *Student* or *Assistant* to verify the compatibility and coherence between different steps of the project. In this way, *User* becomes able to realize his mistakes at any step of the project developed. At the same time, *Assistant* becomes able to check the project that *Student* has developed and observe the incompatibility between different steps of the project.

TraceEngine

TraceEngine enables Student to check the existence of signal on any of the *Wire* or *BreadBoardSocket* components on the *BreadBoard*. *Student* clicks on a point on these components and view the existence of the signal on them.

TraceLed

A *Led* that is used to check the existence of signal on the point clicked by the *Student*. If the point clicked by the *Student* has signal, the *TraceLed* turns ON. Otherwise, it turns OFF.

HelpDataFetcher

Fetches the help documents that are stored locally or online on the web page of BBSim. *HelpDataFetcher* gets these help documents into the program and converts them into a viewable format.

PrintFormatter

PrintFormatter converts any of the project components into a printable format whenever the *User* demands *BBSim* to print them. These components are Truth Table, Karnaugh Map, Circuit Schematic, Circuit on Breadboard, and the result of Simulation.

PrintProperties

PrintProperties are the paper size, page orientation and number of copies for printing. By default, paper size is A4, page orientation is portrait and number of copies is 1. However, User (*Student* or *Assistant*) can change them as he desires. If *User* do not make any changes, default properties is used while printing.

PrintStatus

PrintStatus stores the status of the print jobs that the *Student* or *Assistant* has demanded. There are 5 different status options for every print job; "Formatting.", "Printing.", "Canceled.", "Printed." and "Error.".

4.1.2. Class Diagrams

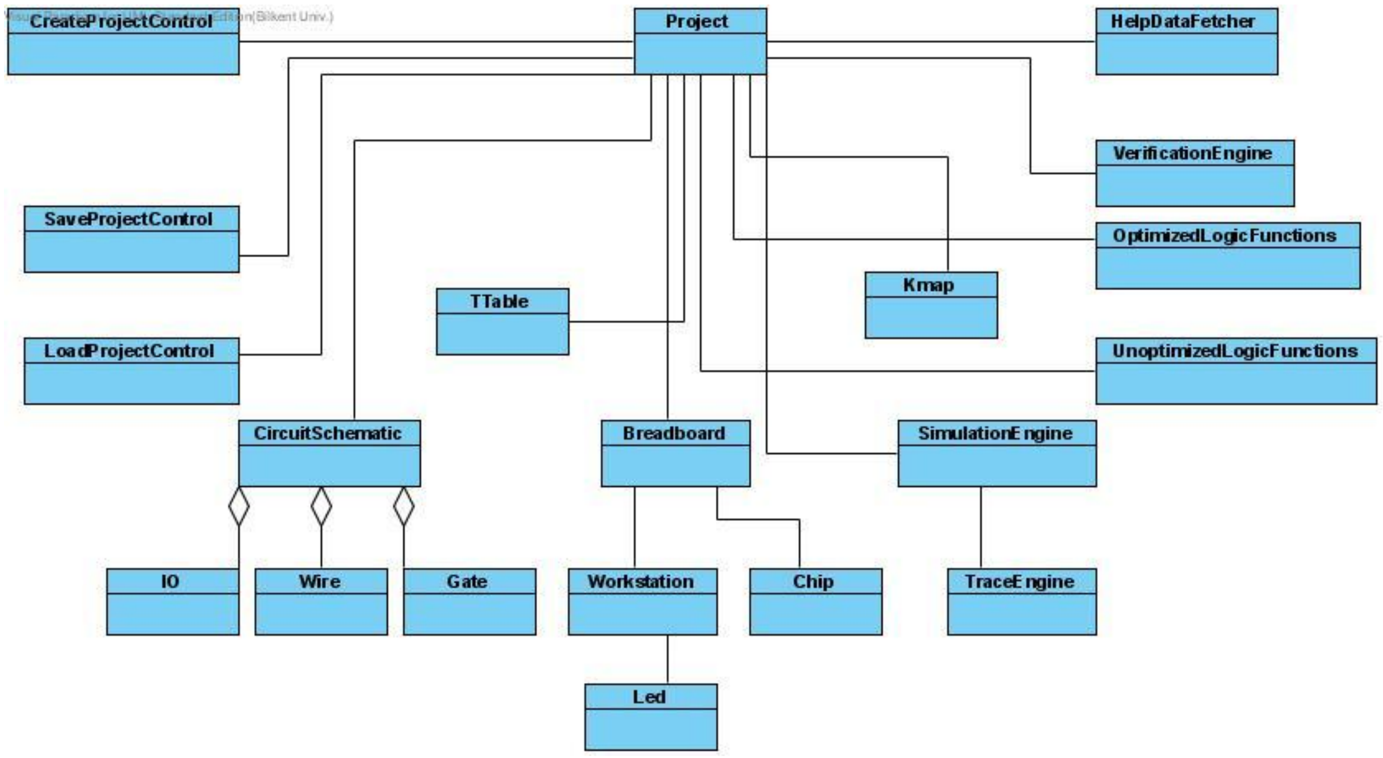


Figure 6: Class Diagram of Overall System

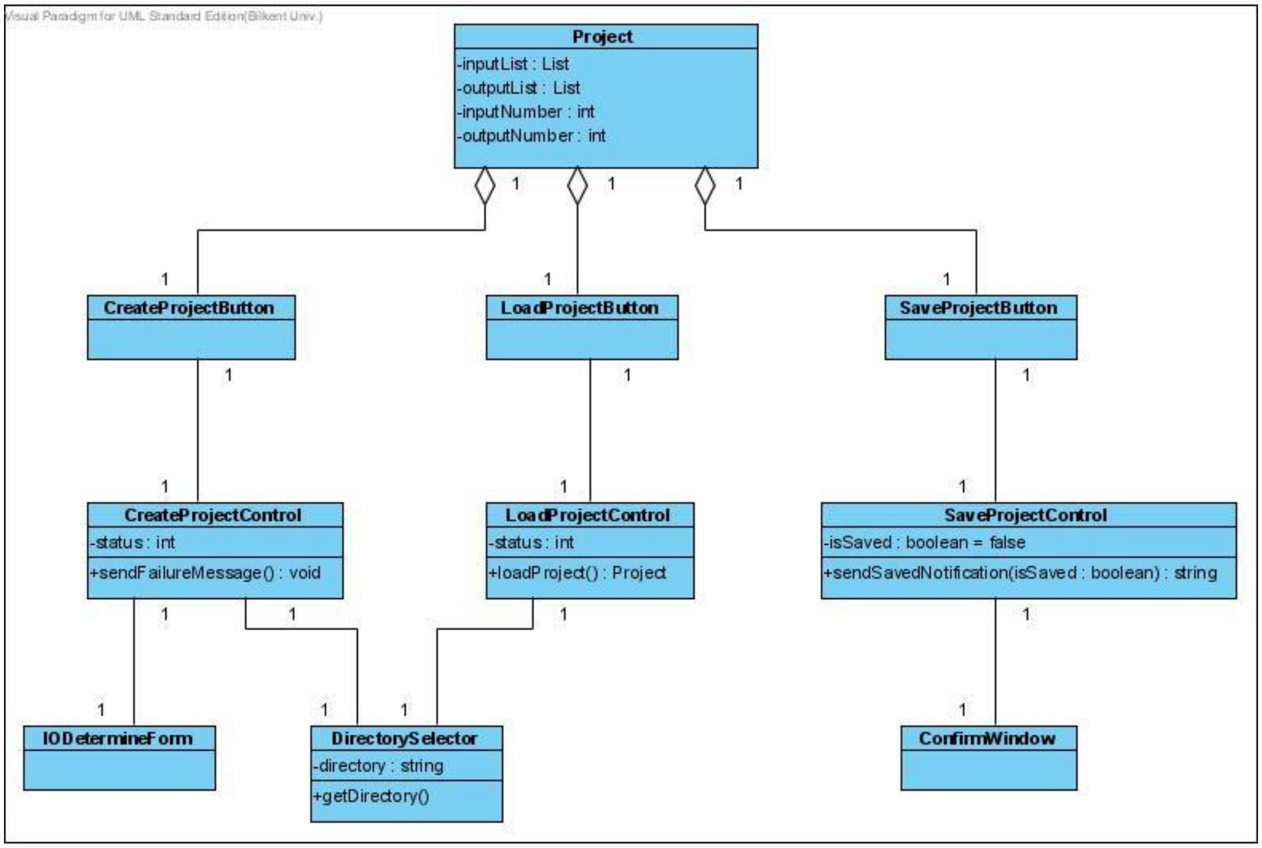


Figure 7 - Class Diagram of “Manage Project”

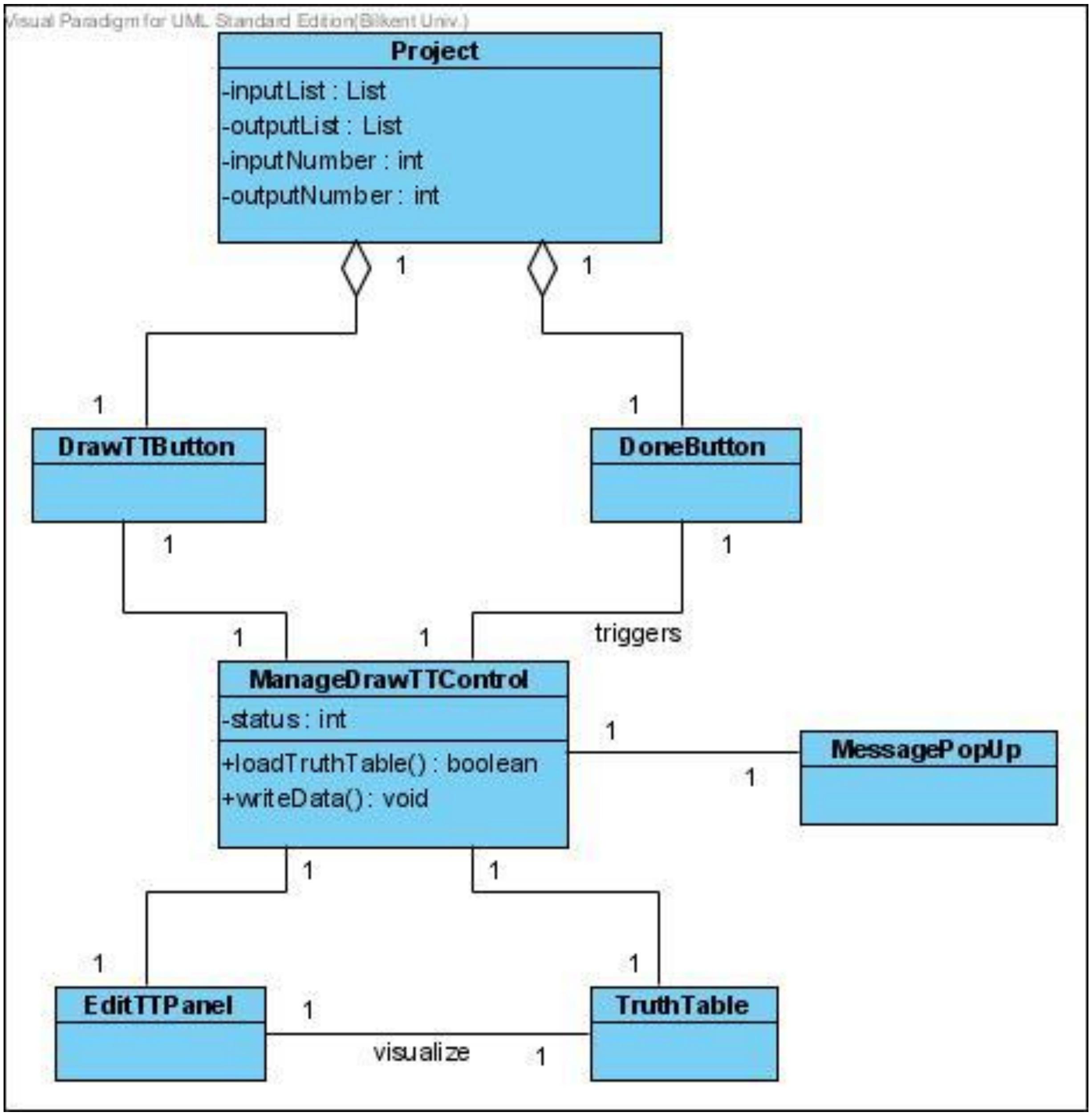


Figure 8 - Class Diagram of "Draw Truth Table"

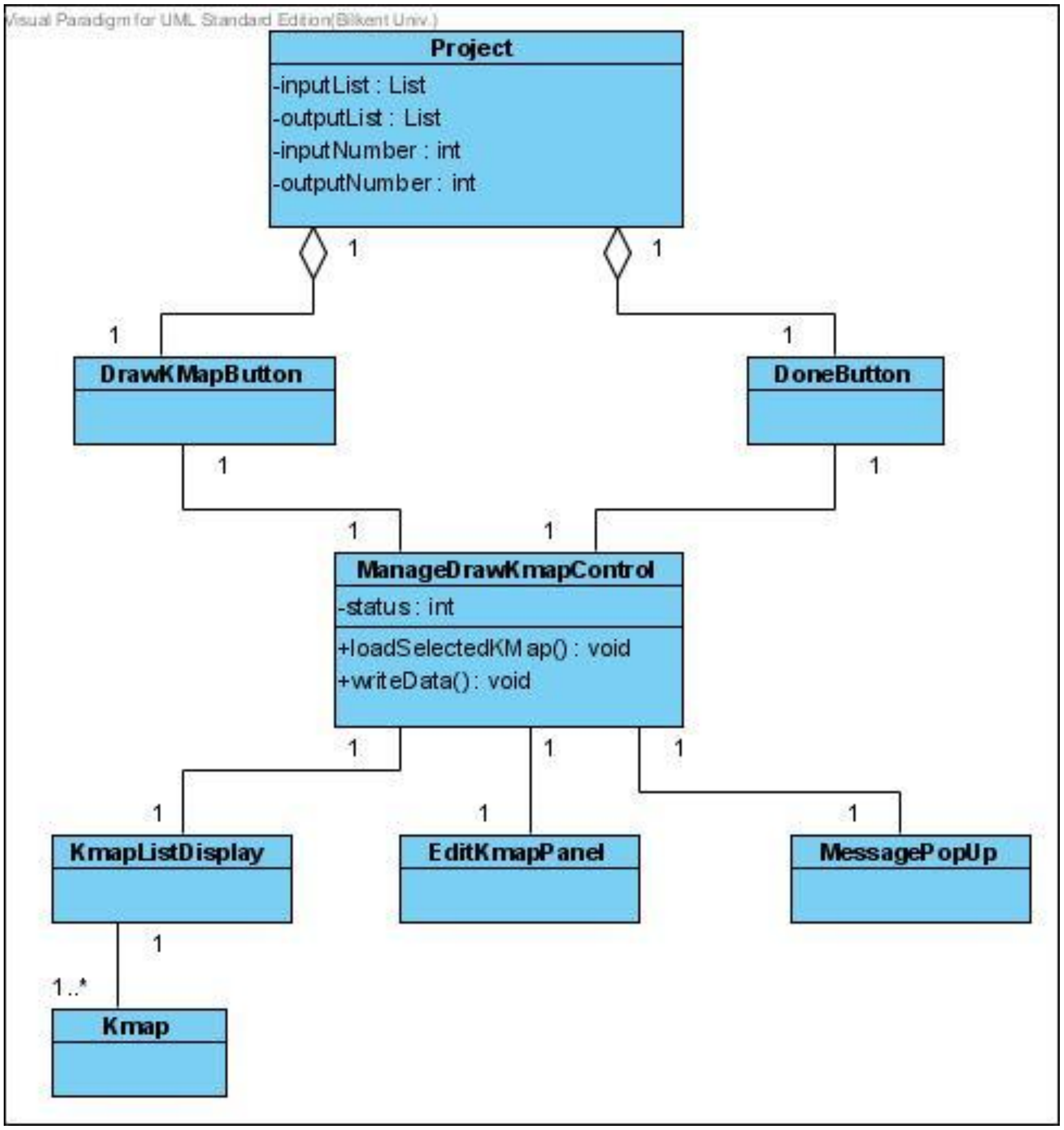


Figure 9 - Class Diagram of “Draw Karnaugh Map”

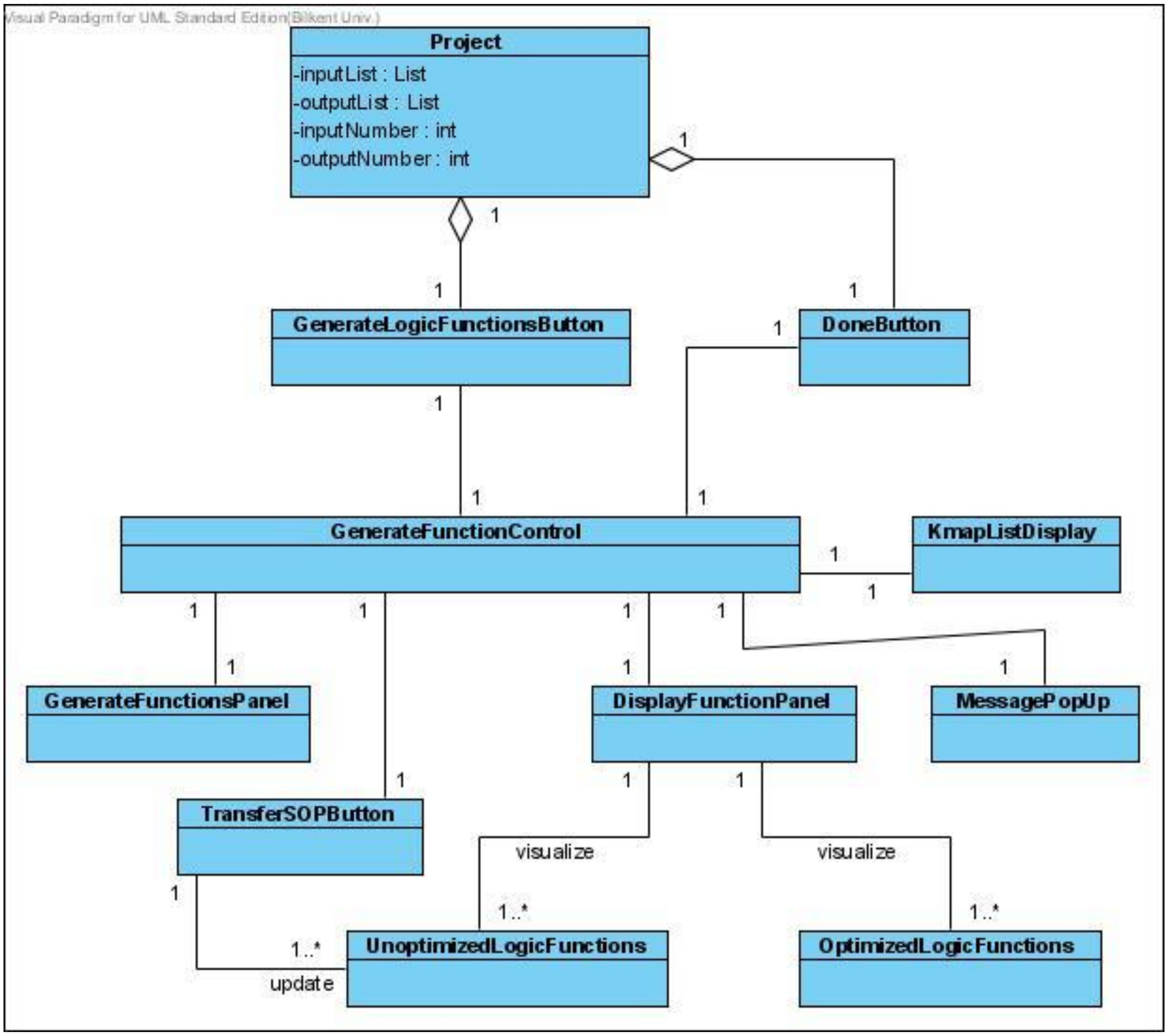


Figure 10 - Class Diagram of "Generate Logic Functions"

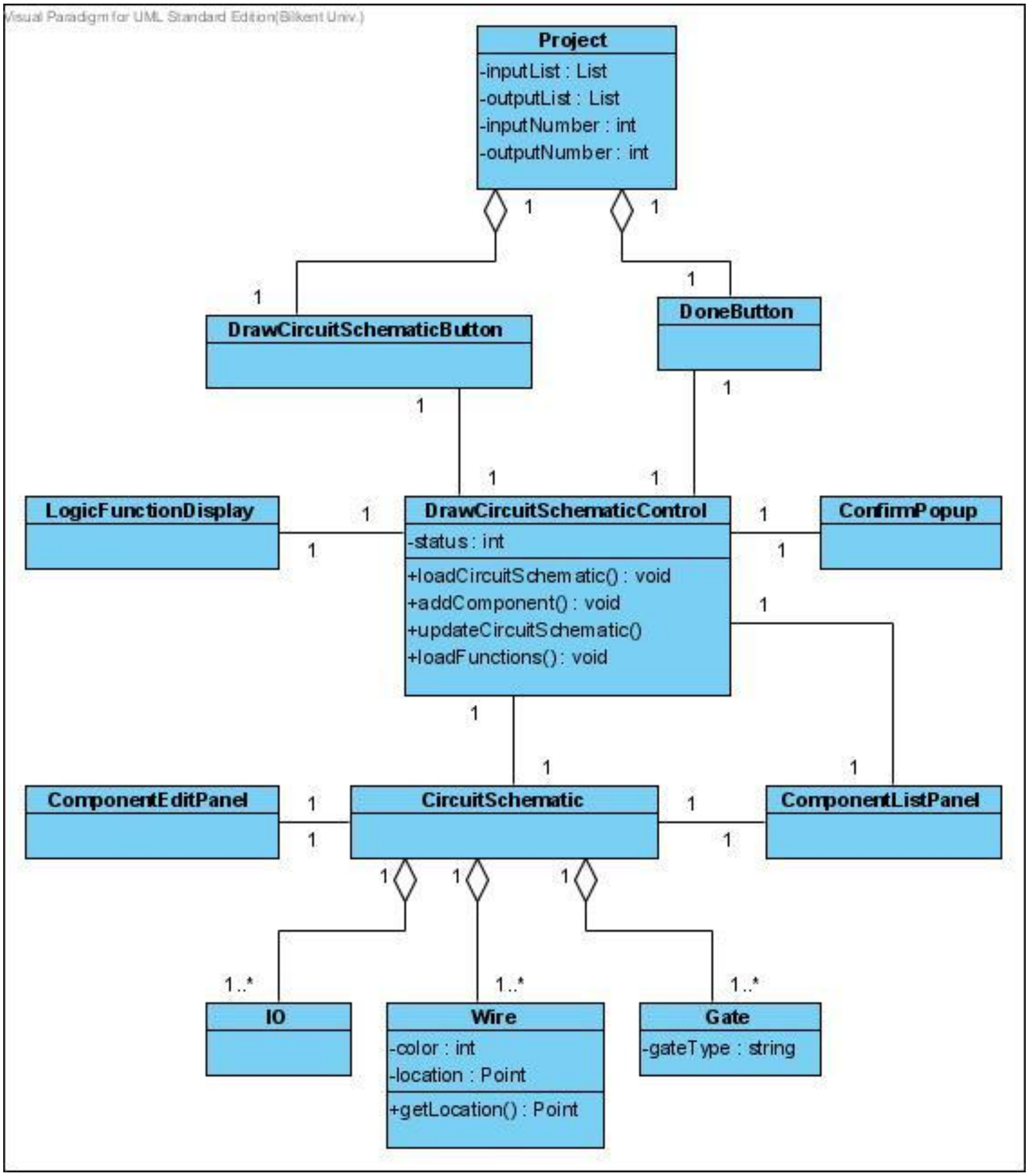


Figure 11 - Class Diagram of "Draw Circuit Schematic"

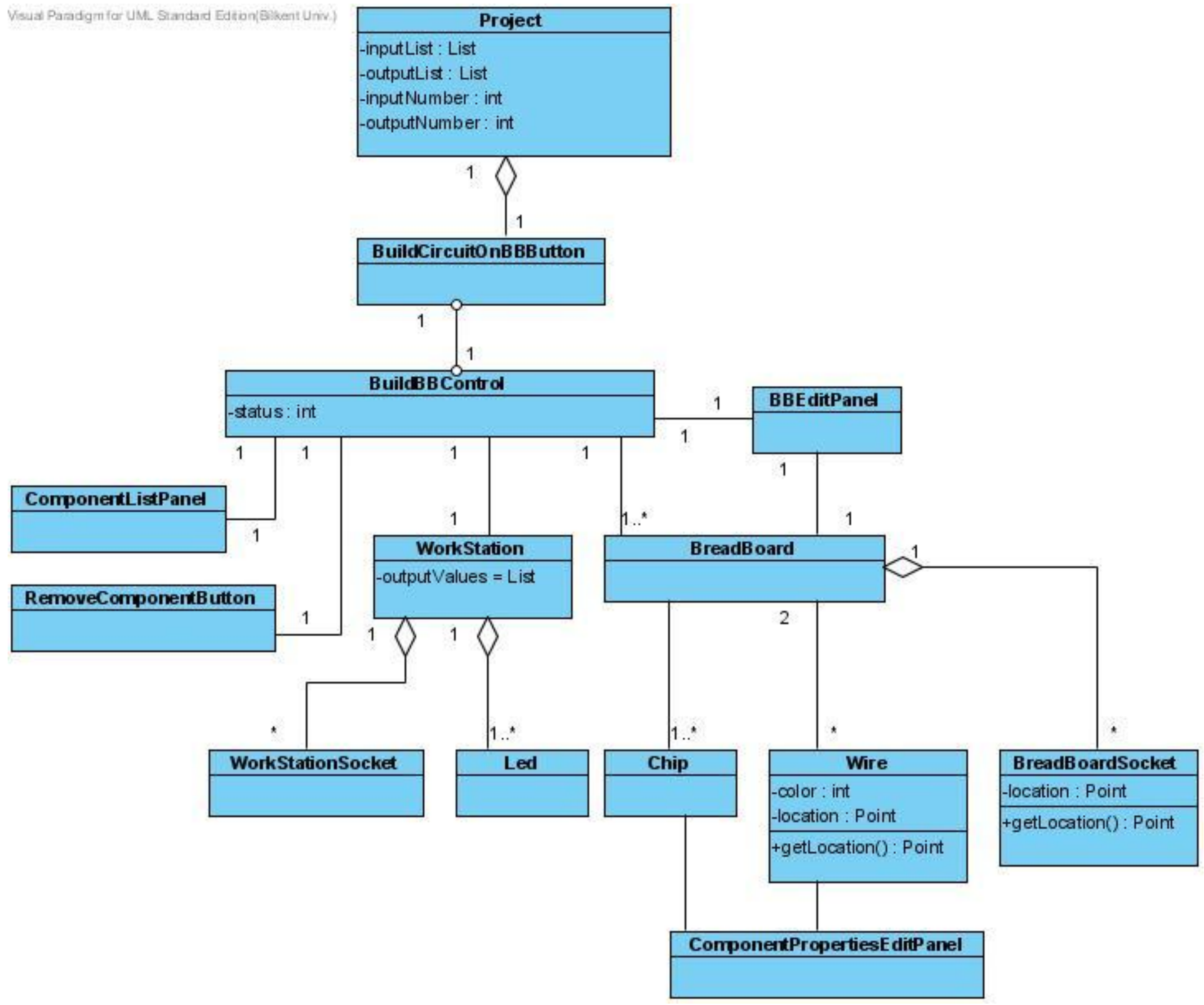


Figure 12 - Objects of “Build Circuit”

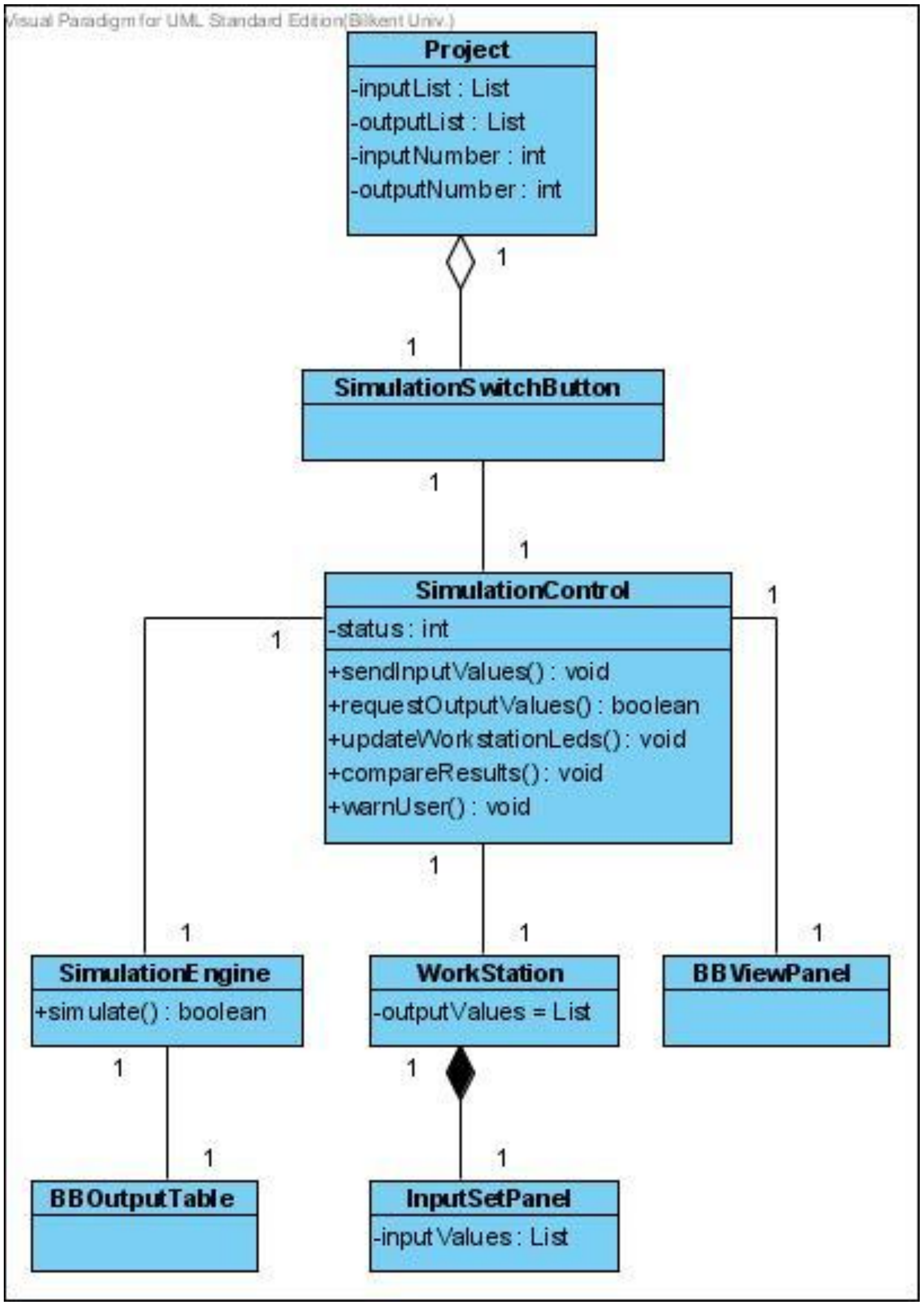


Figure 13 - Class Diagram of "Simulate Circuit"

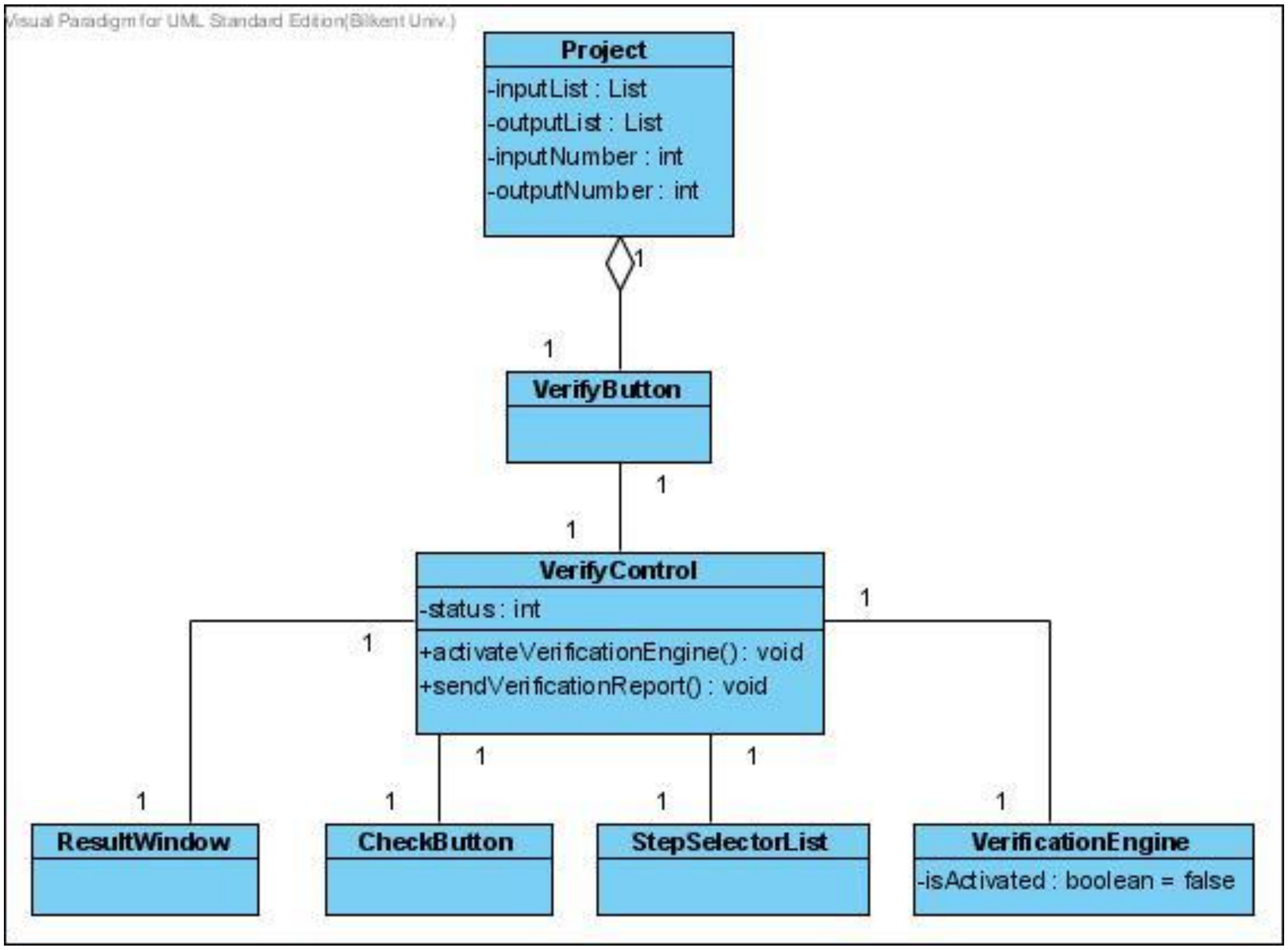


Figure 14 - Class Diagram of "Verify"

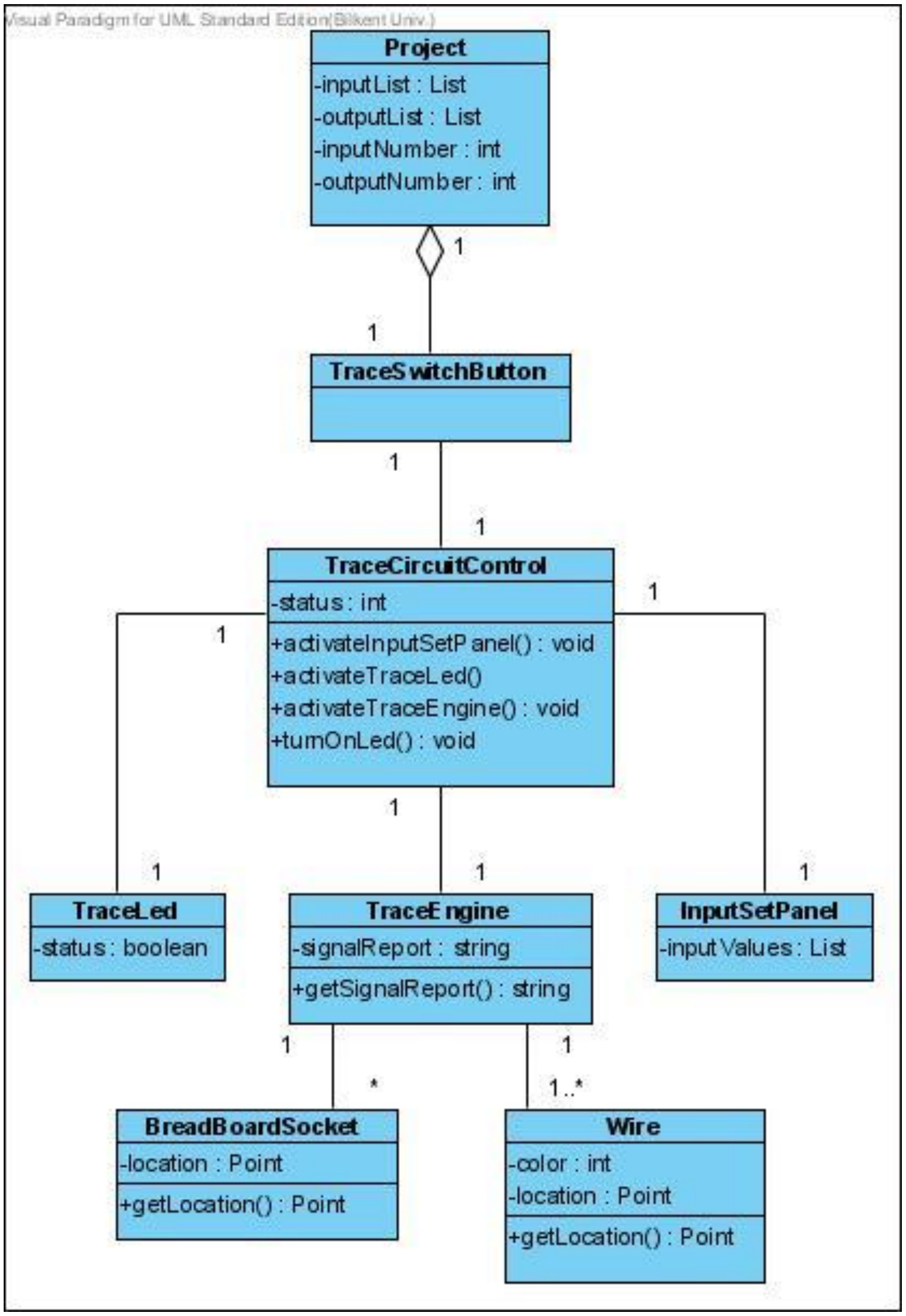


Figure 15 - Class Diagram of "Trace Circuit"

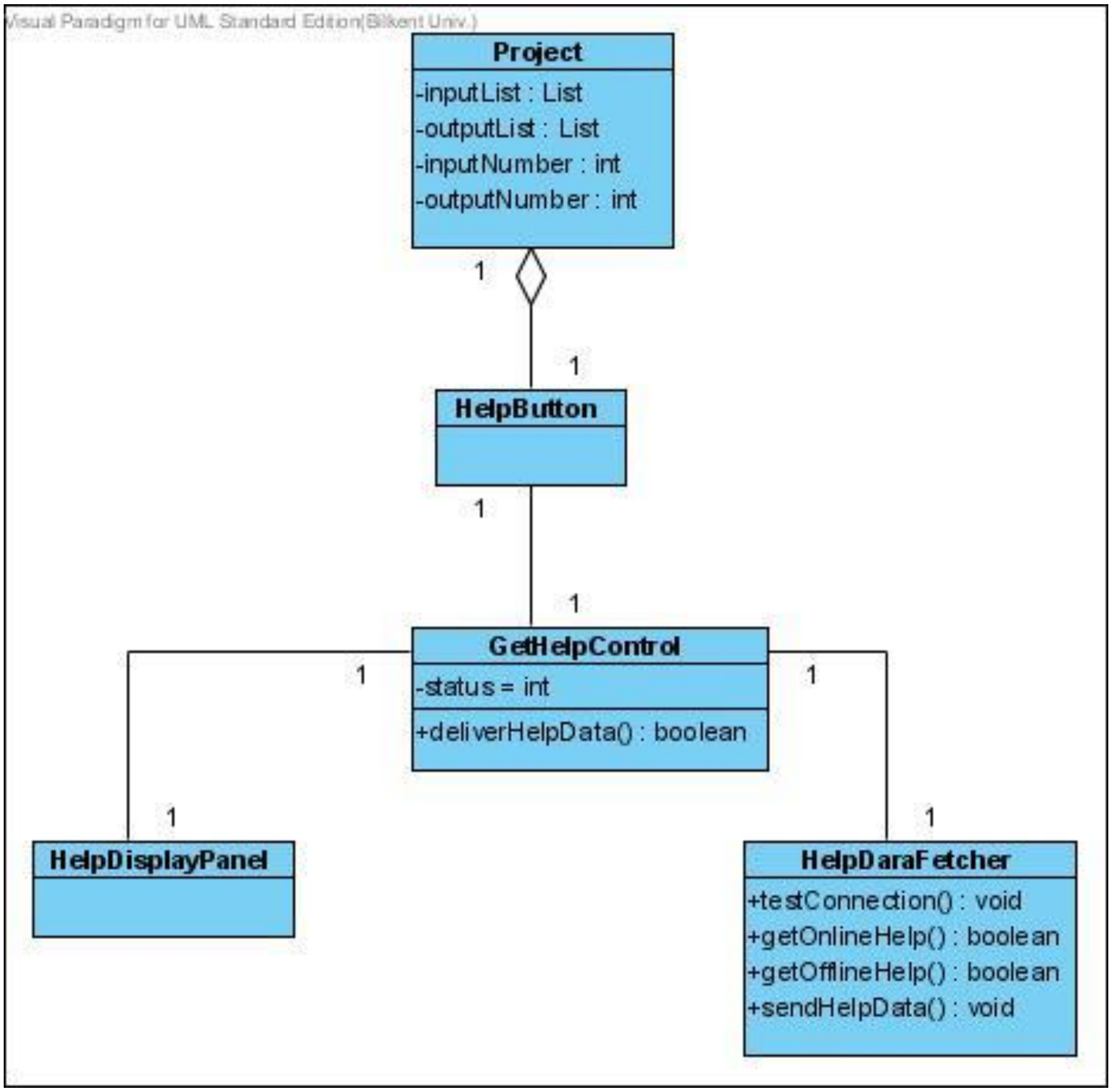


Figure 16 - Class Diagram of "Get Help"

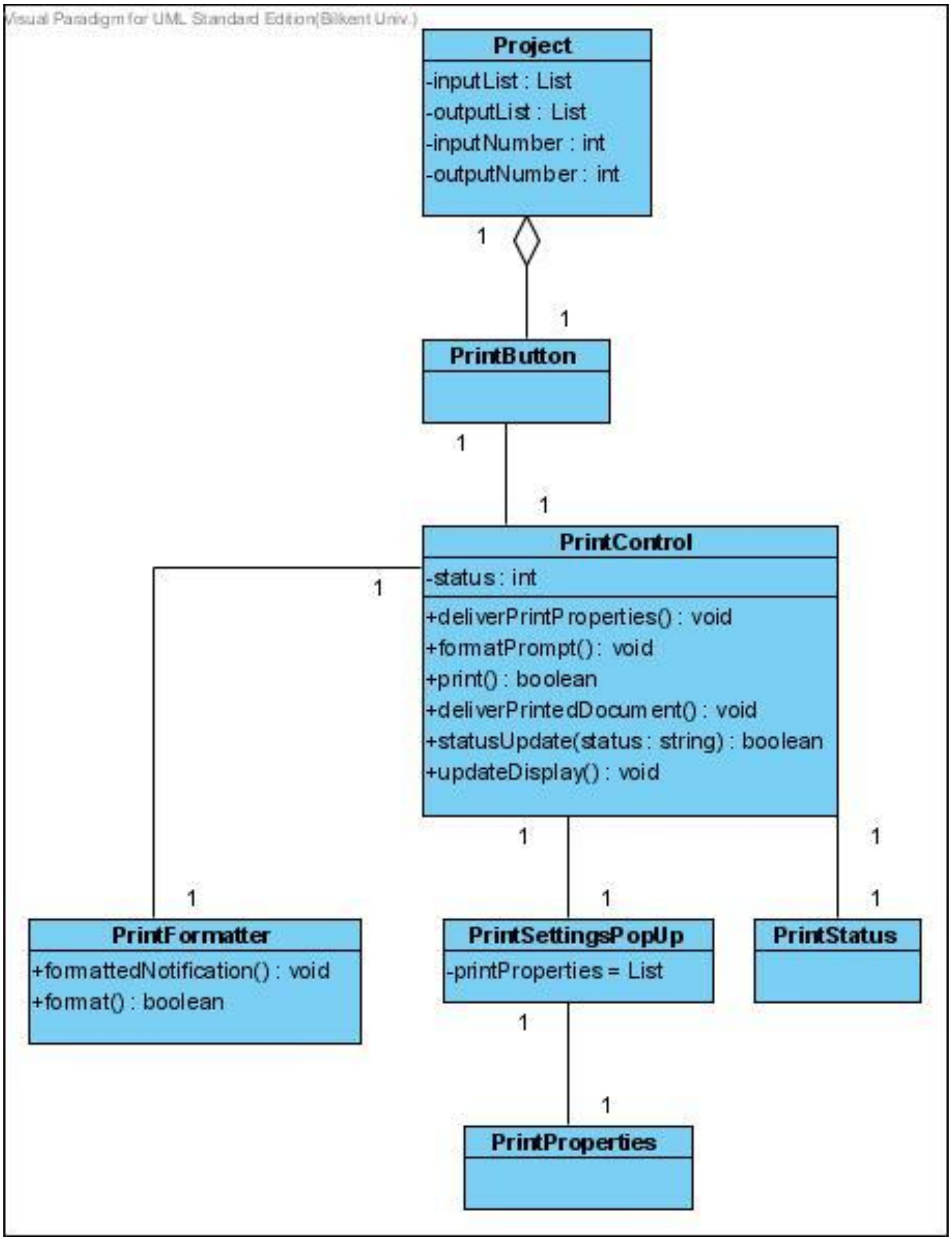


Figure 17 - Class Diagram of "Print"

4.2. Dynamic Models

4.2.1. Sequence Diagram

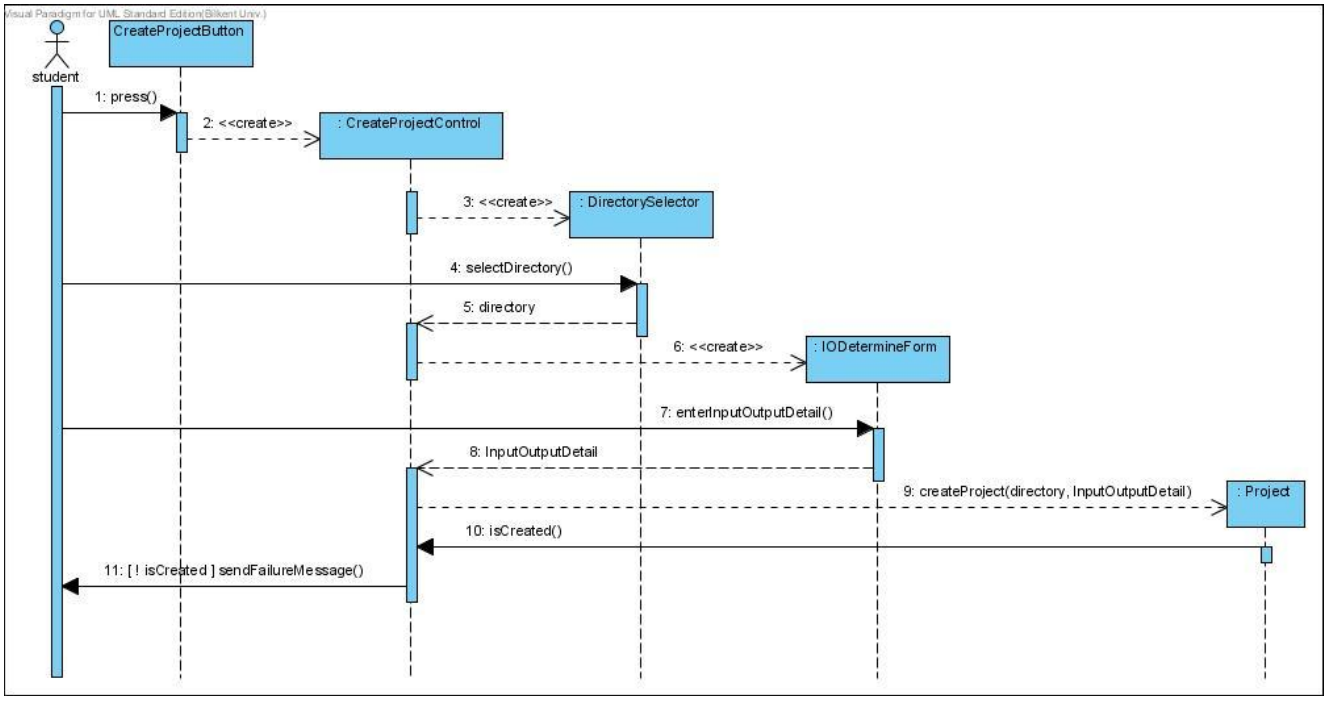


Figure 18 - Sequence Diagram for "Create Project"

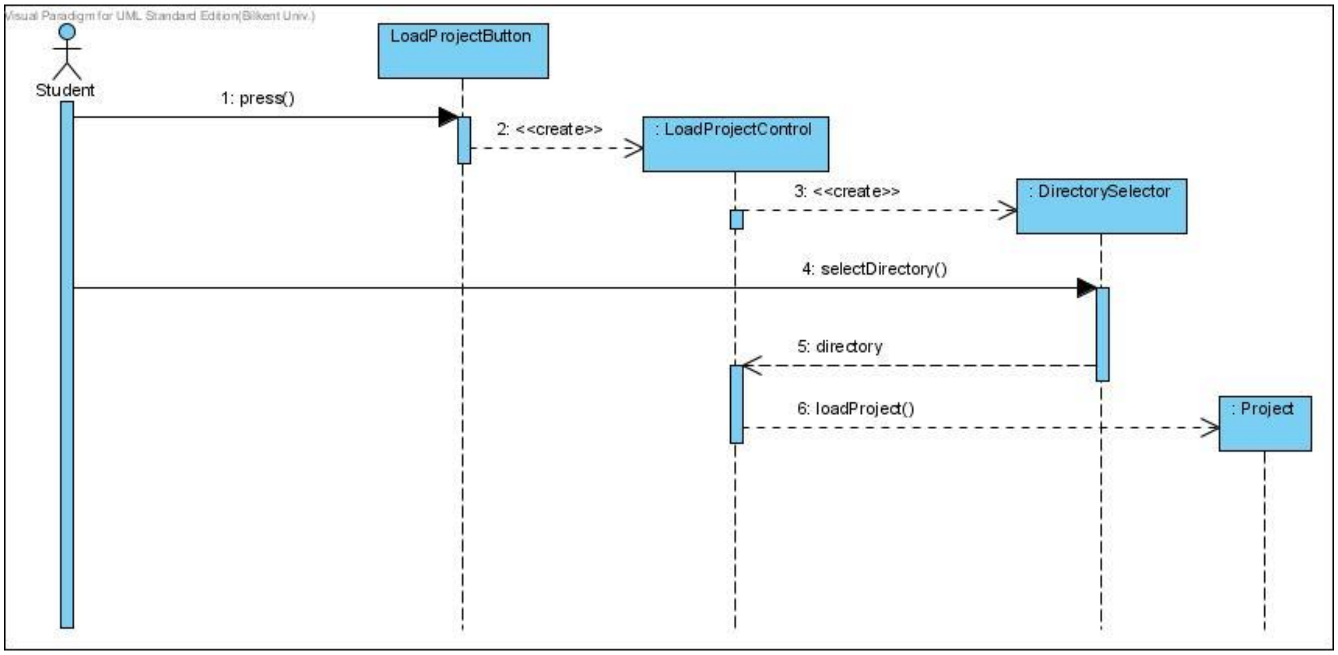


Figure 19 - Sequence Diagram for "Load Project"

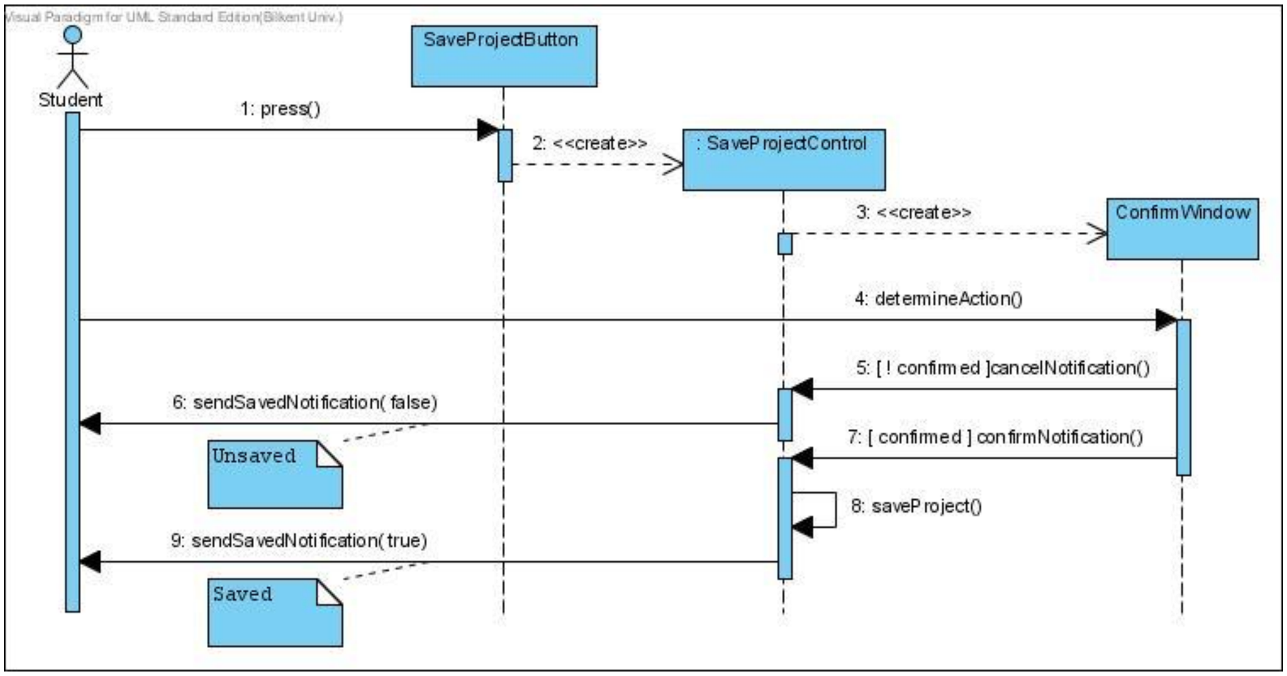

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Student
SaveProjectButton
: SaveProjectControl
ConfirmWindow
1: press()
2: <<create>>
3: <<create>>
4: determineAction()
5: [! confirmed]cancelNotification()
6: sendSavedNotification(false)
Unsaved
7: [confirmed] confirmNotification()
8: saveProject()
9: sendSavedNotification(true)
Saved

Figure 20 - Sequence Diagram for "Save Project"

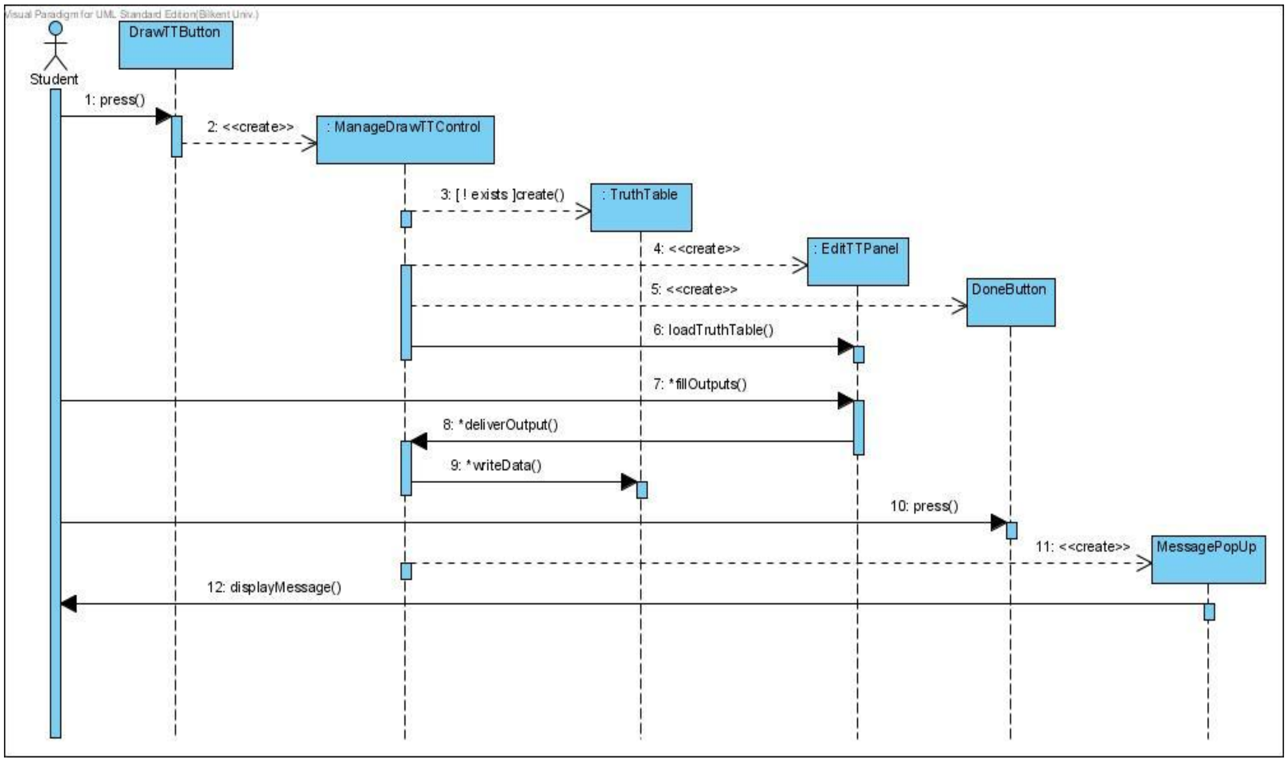


Figure 21 - Sequence Diagram for "Draw Truth Table"

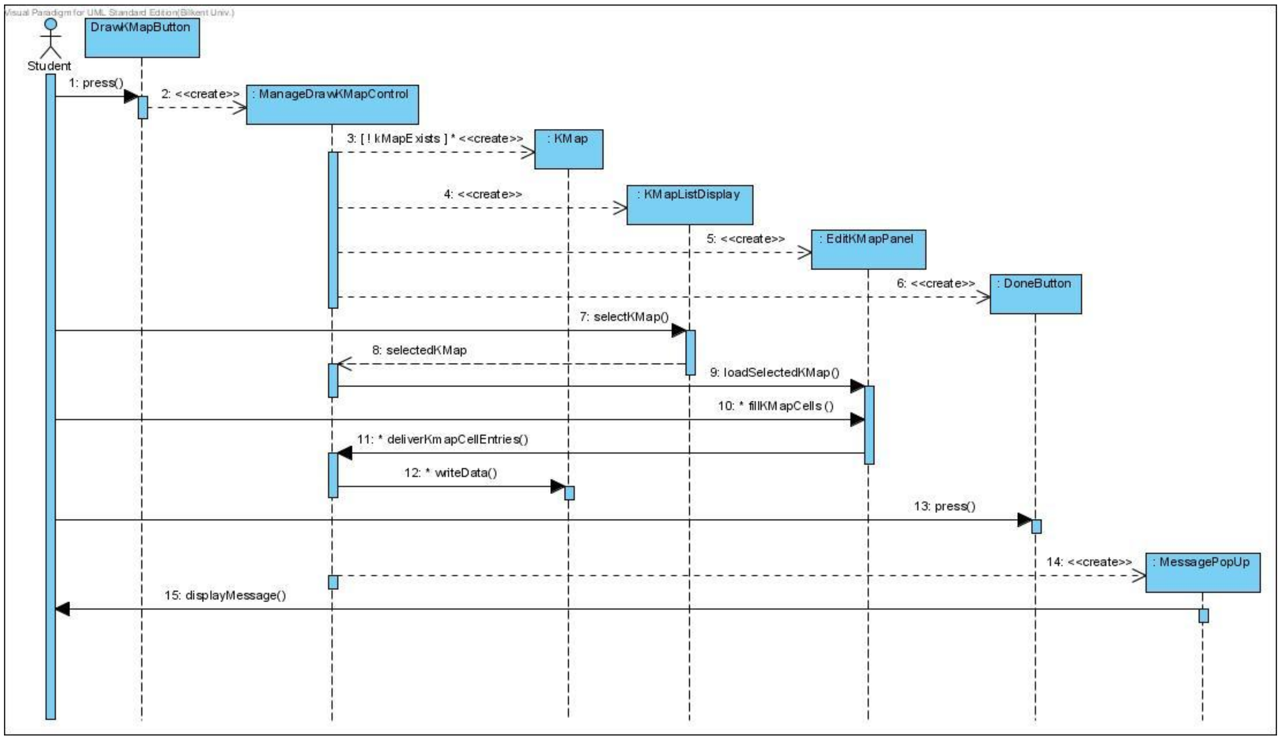


Figure 22 - Sequence Diagram for "Draw Karnaugh Map"

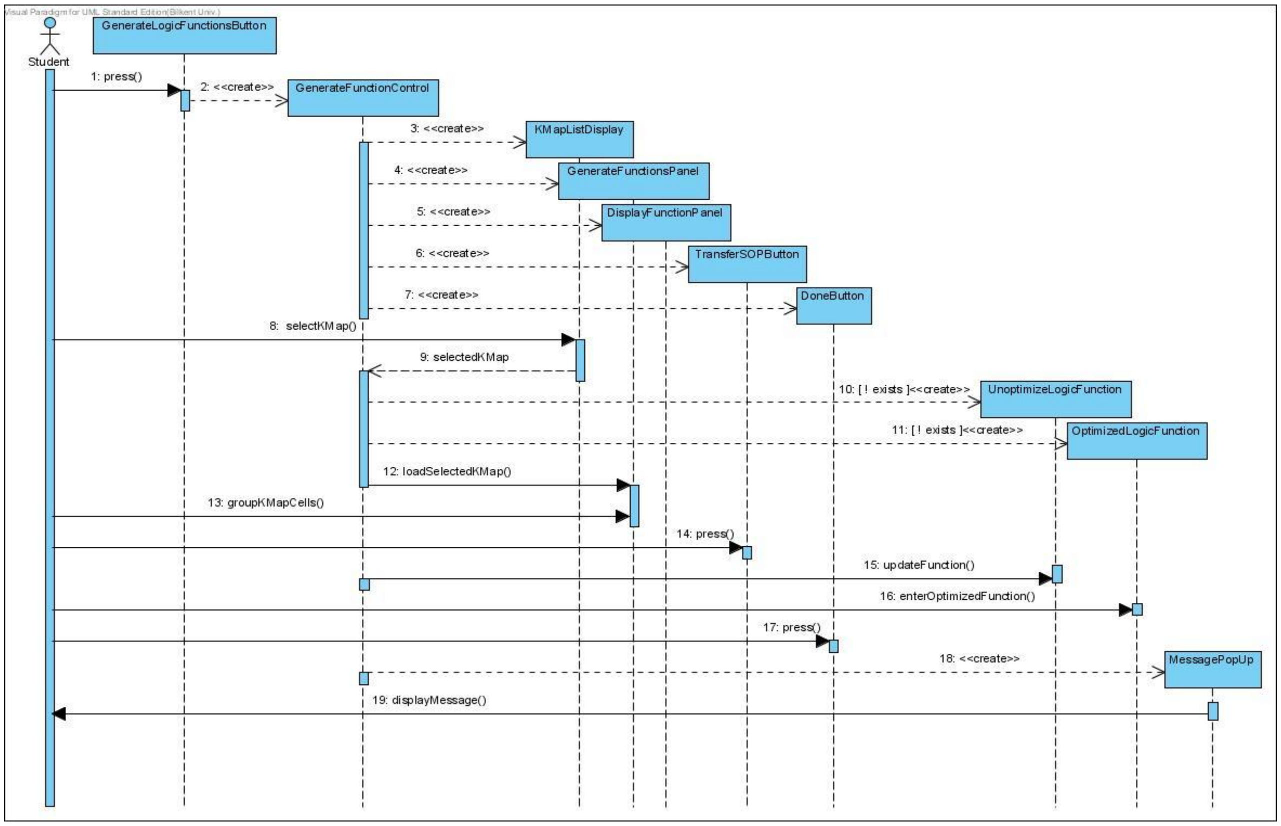


Figure 23 - Sequence Diagram for "Generate Logic Functions"

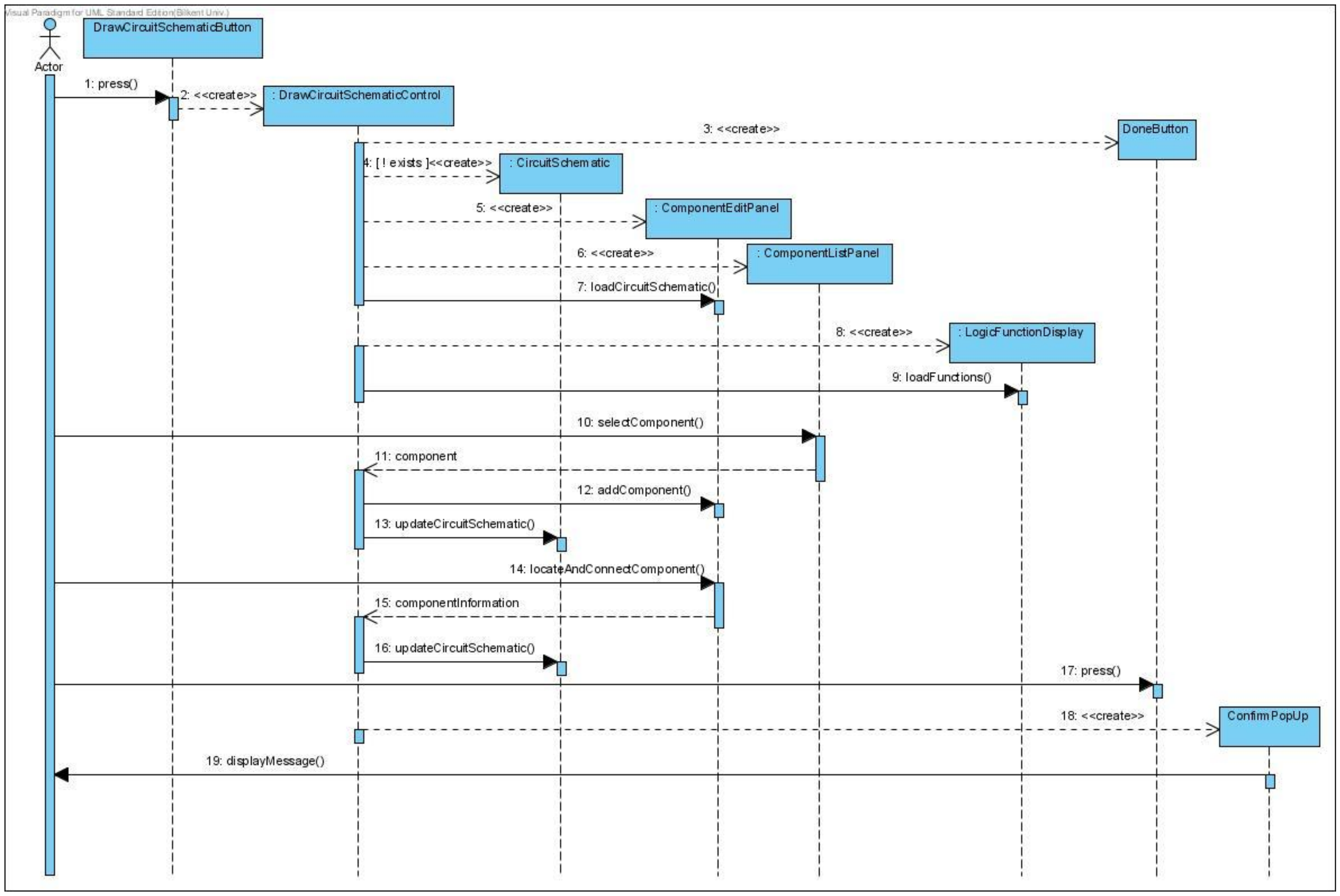


Figure 24 - Sequence Diagram for "Draw Circuit Schematic"

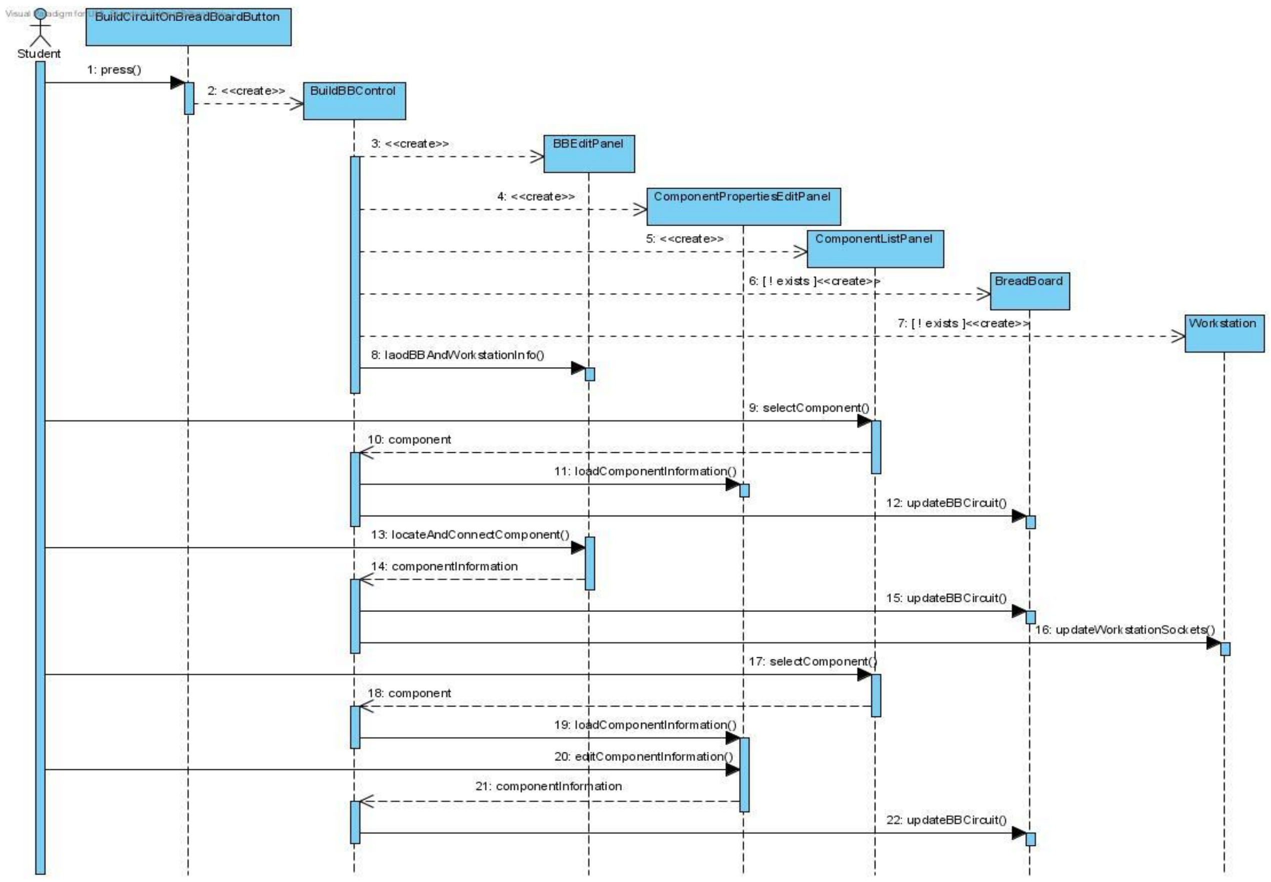


Figure 25 - Sequence Diagram for "Build circuit on Breadboard"

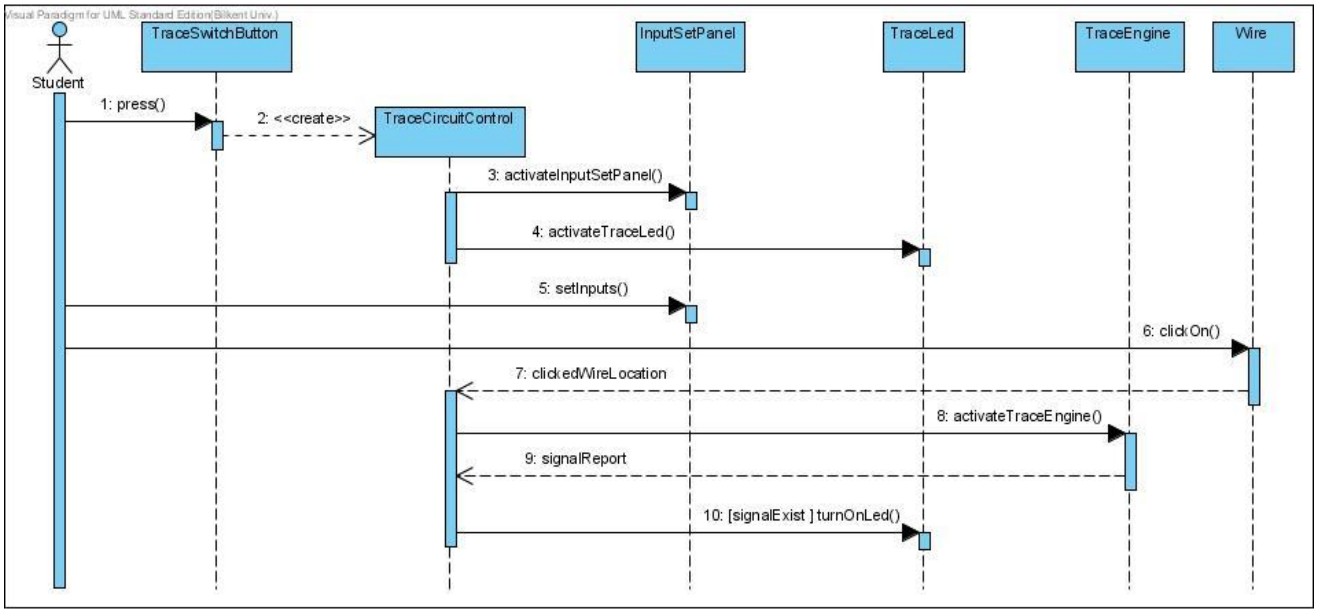


Figure 26 - Sequence Diagram for "Trace Circuit"

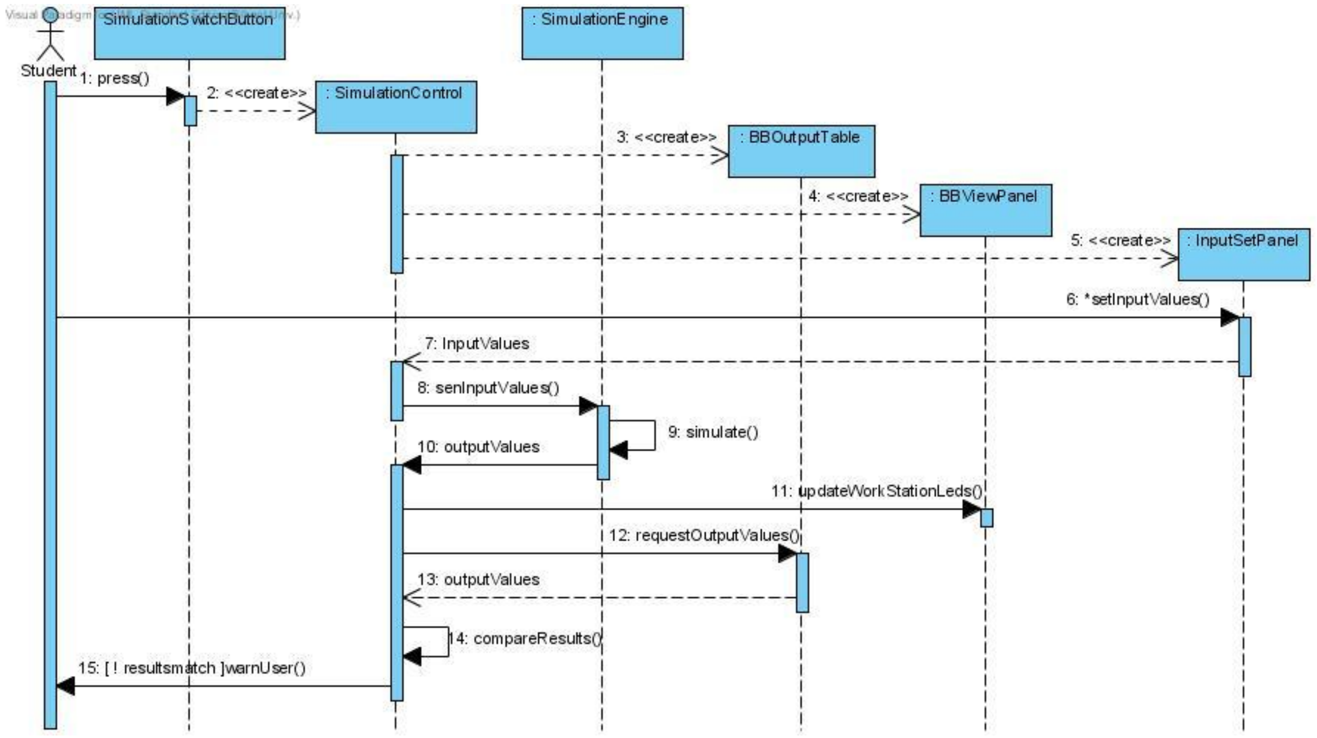


Figure 27 - Sequence Diagram for "Simulate Circuit"

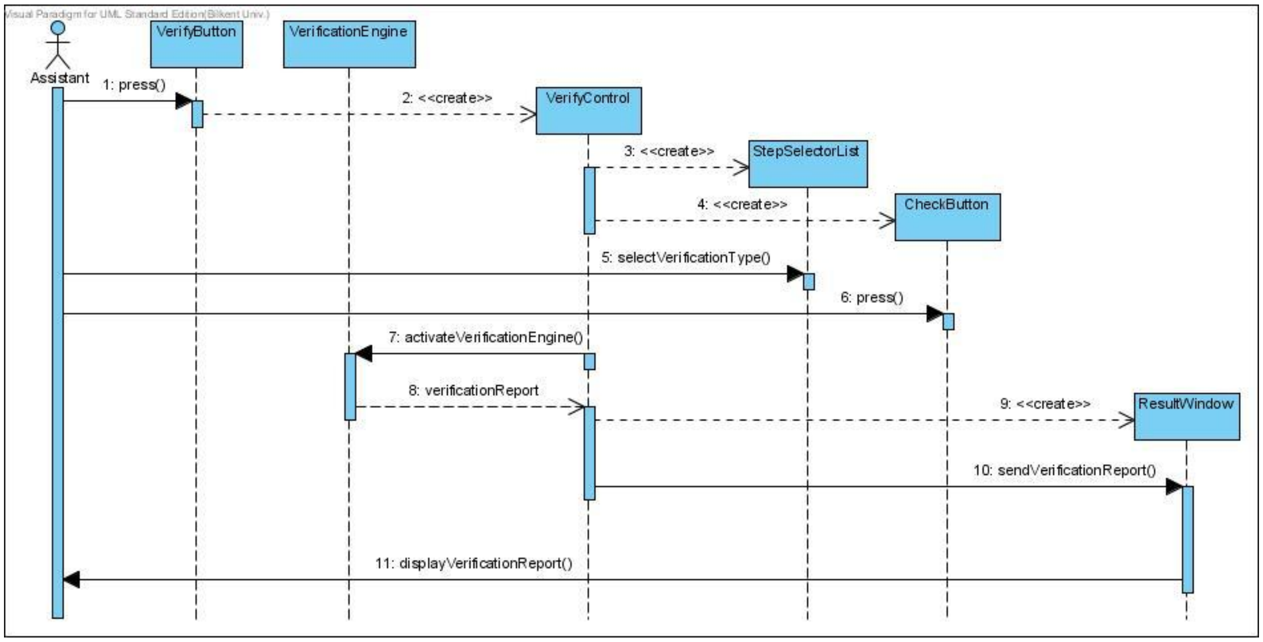


Figure 28 - Sequence Diagram for "Verify"

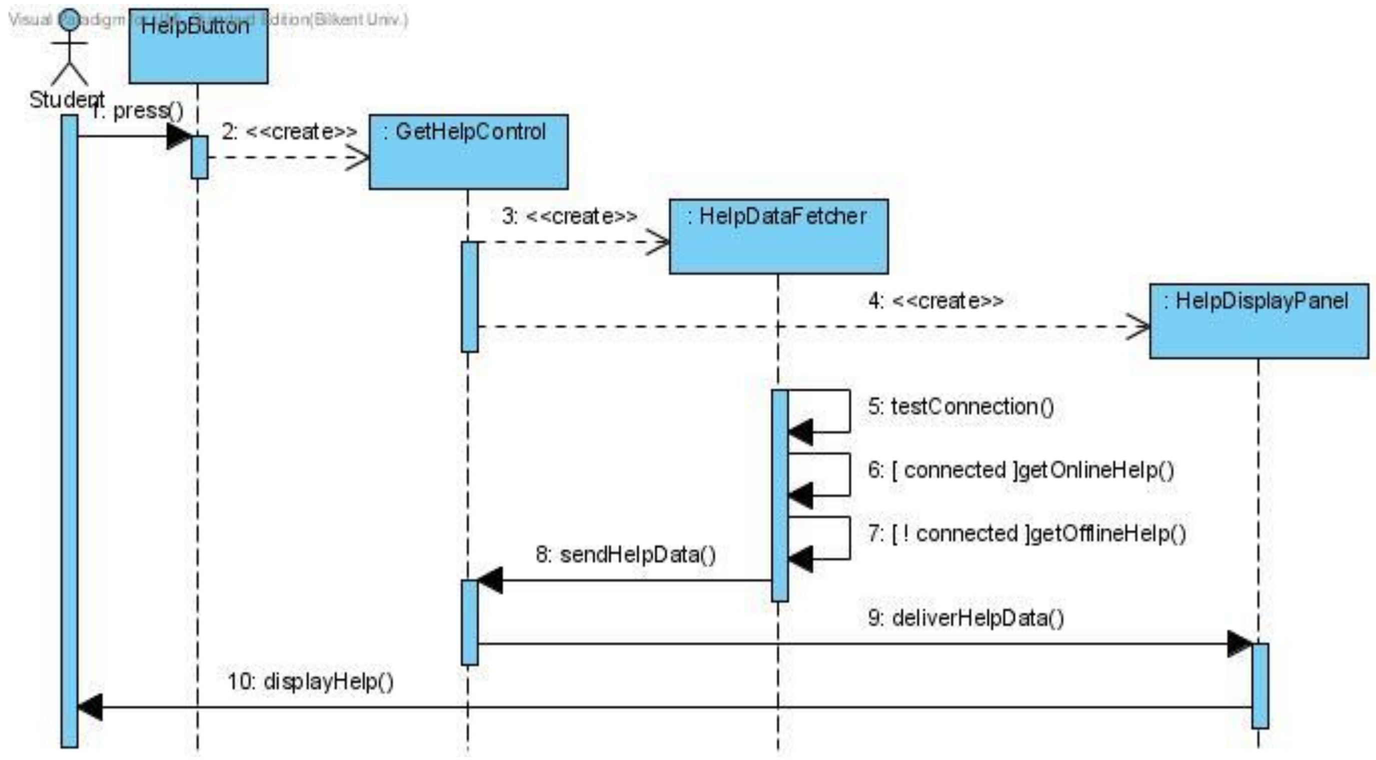


Figure 29 - Sequence Diagram for "Get Help"

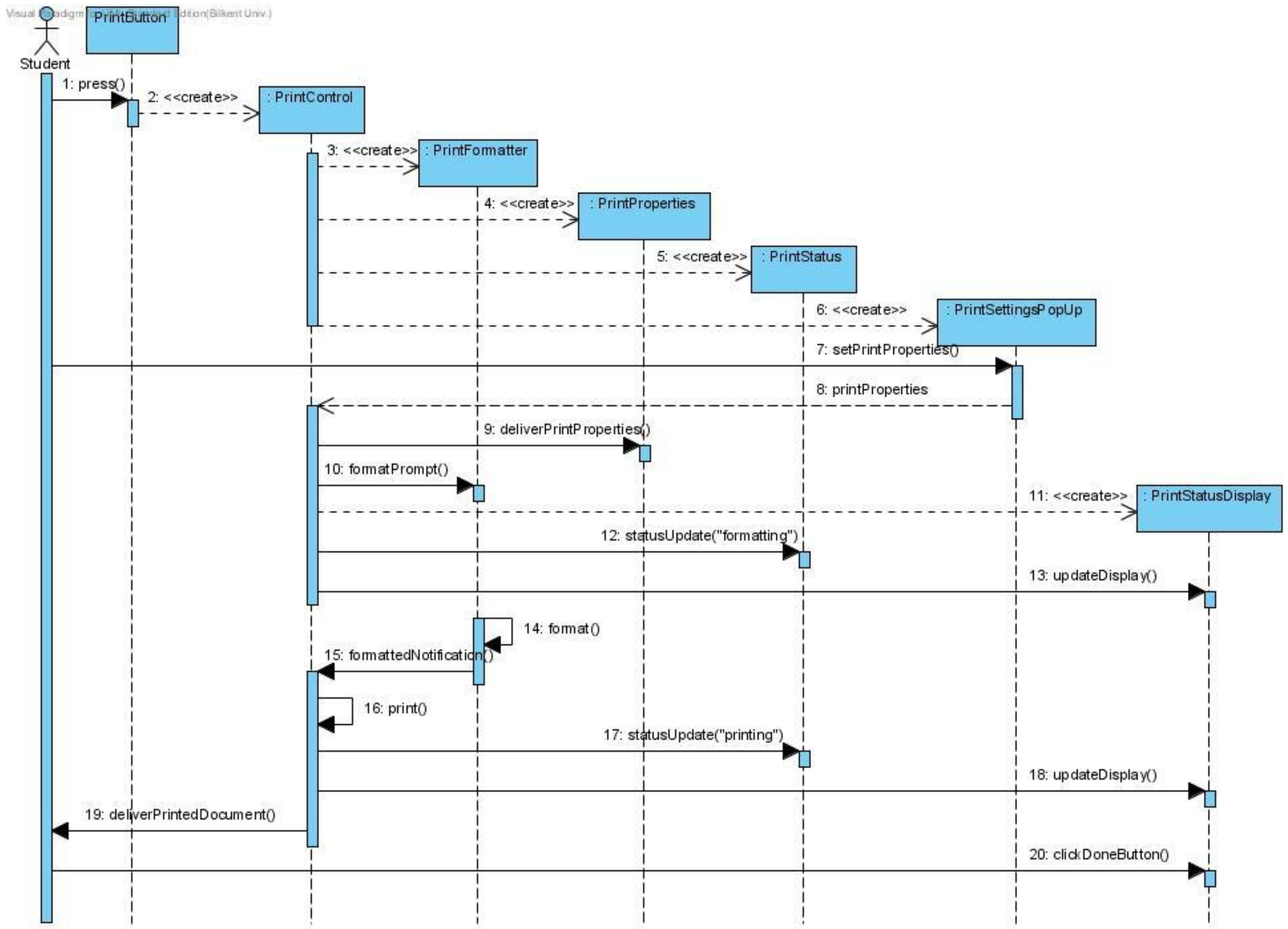
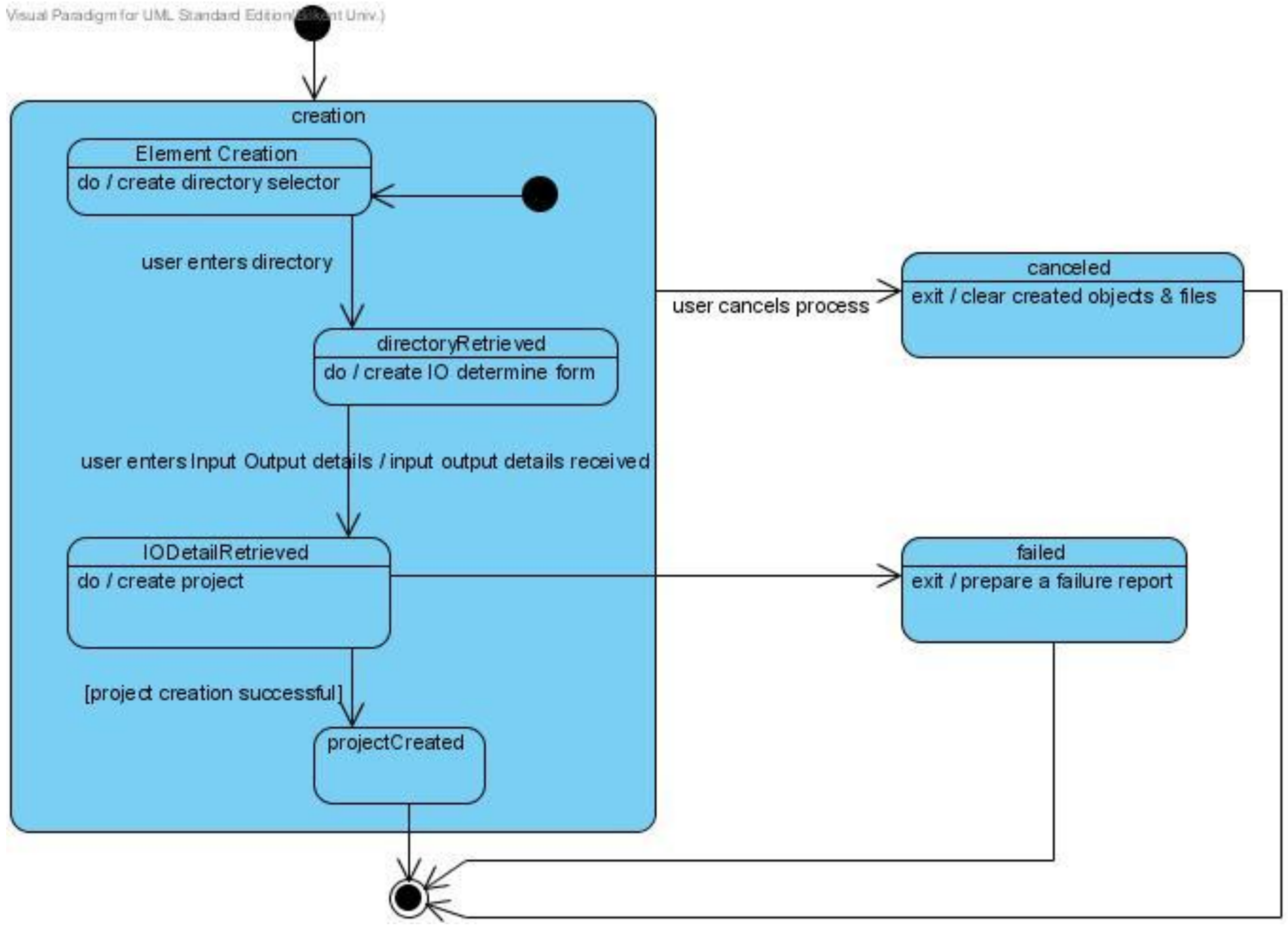


Figure 30 - Sequence Diagram for "Print"

4.2.2. State Chart Diagrams



Visual Paradigm for UML Standard Edition(Bilkent Univ.)
creation
Element Creation
do / create directory selector
user enters directory
directoryRetrieved
do / create IO determine form
user enters Input Output details / input output details received
IODetailRetrieved
do / create project
[project creation successful]
projectCreated
user cancels process
canceled
exit / clear created objects & files
failed
exit / prepare a failure report

Figure 31 - State Chart for "CreateProjectControl"

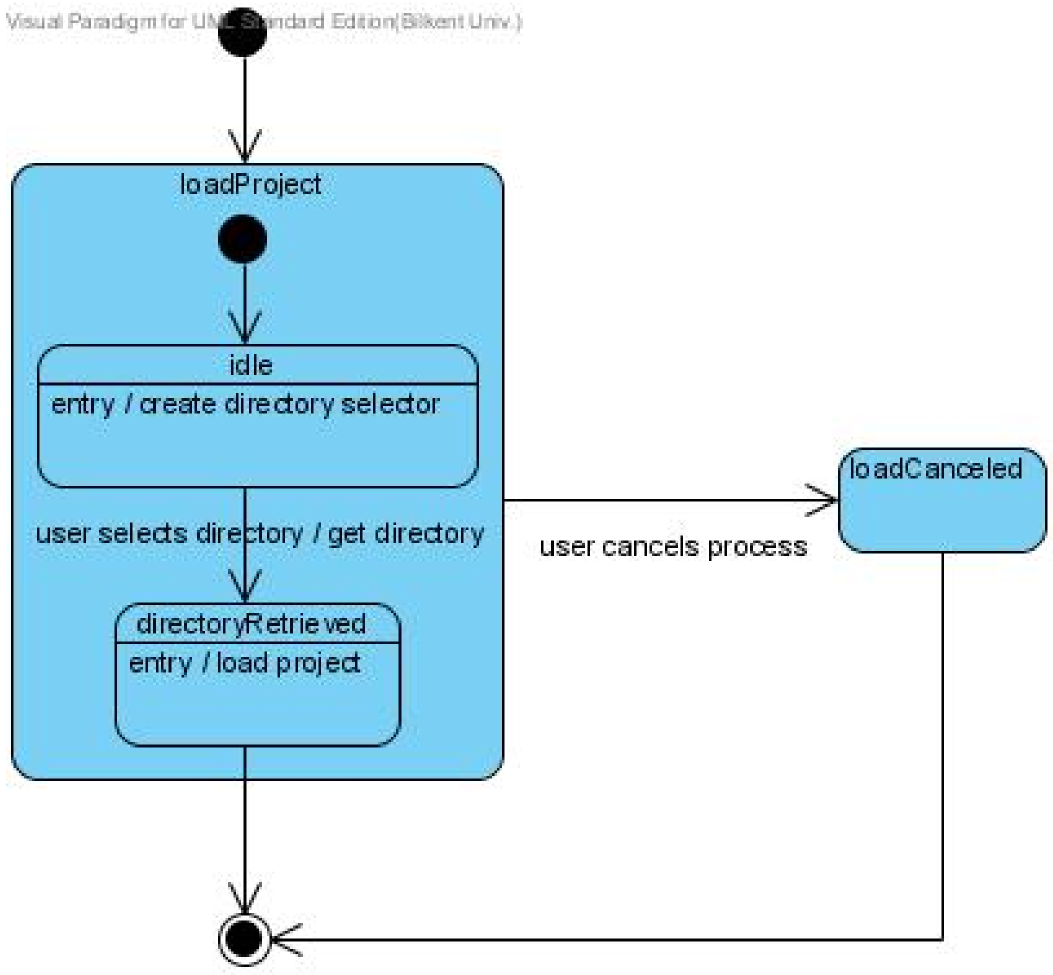

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
loadProject
idle
entry / create directory selector
user selects directory / get directory
directoryRetrieved
entry / load project
loadCanceled
user cancels process

Figure 32 - State Chart for "LoadProjectControl"

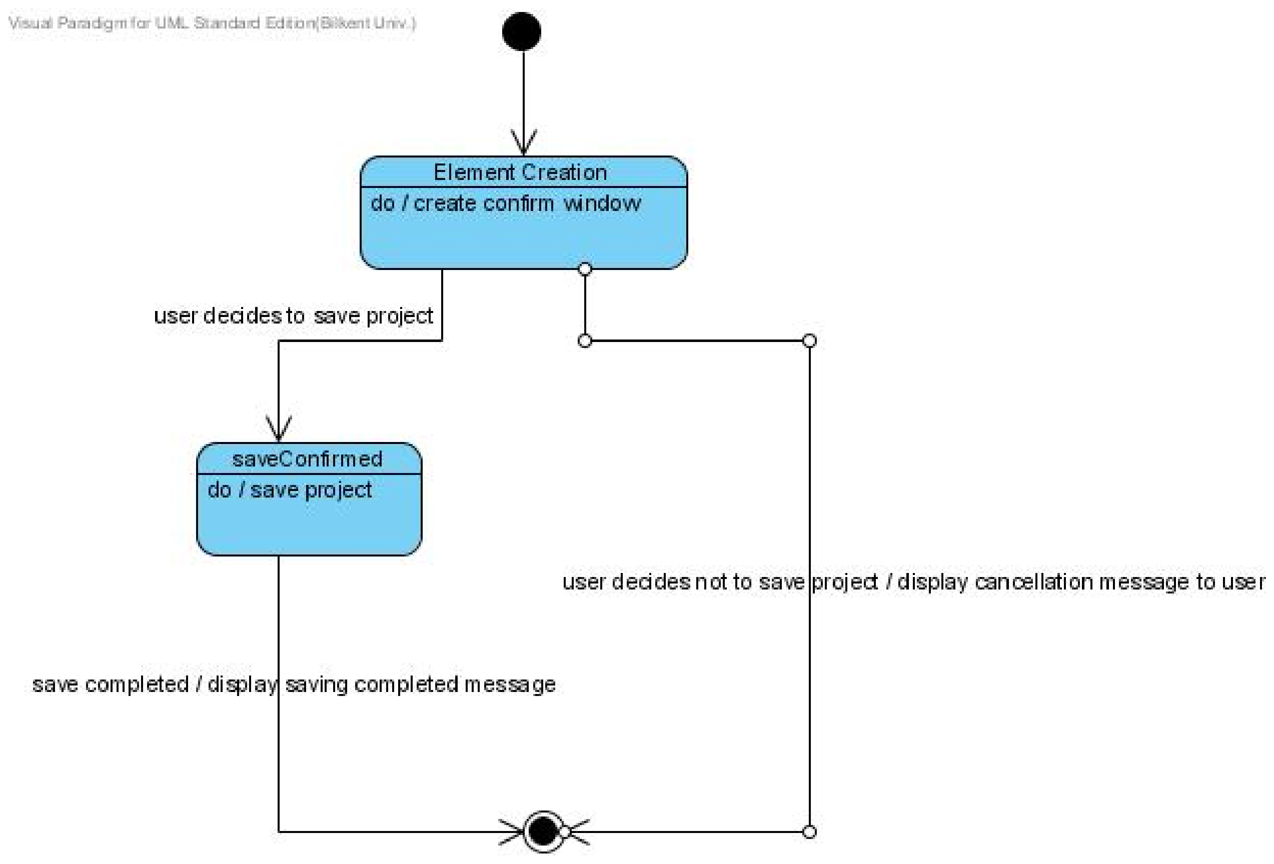

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Element Creation
do / create confirm window
user decides to save project
saveConfirmed
do / save project
user decides not to save project / display cancellation message to user
save completed / display saving completed message

Figure 33 - State Chart for "SaveProjectControl"

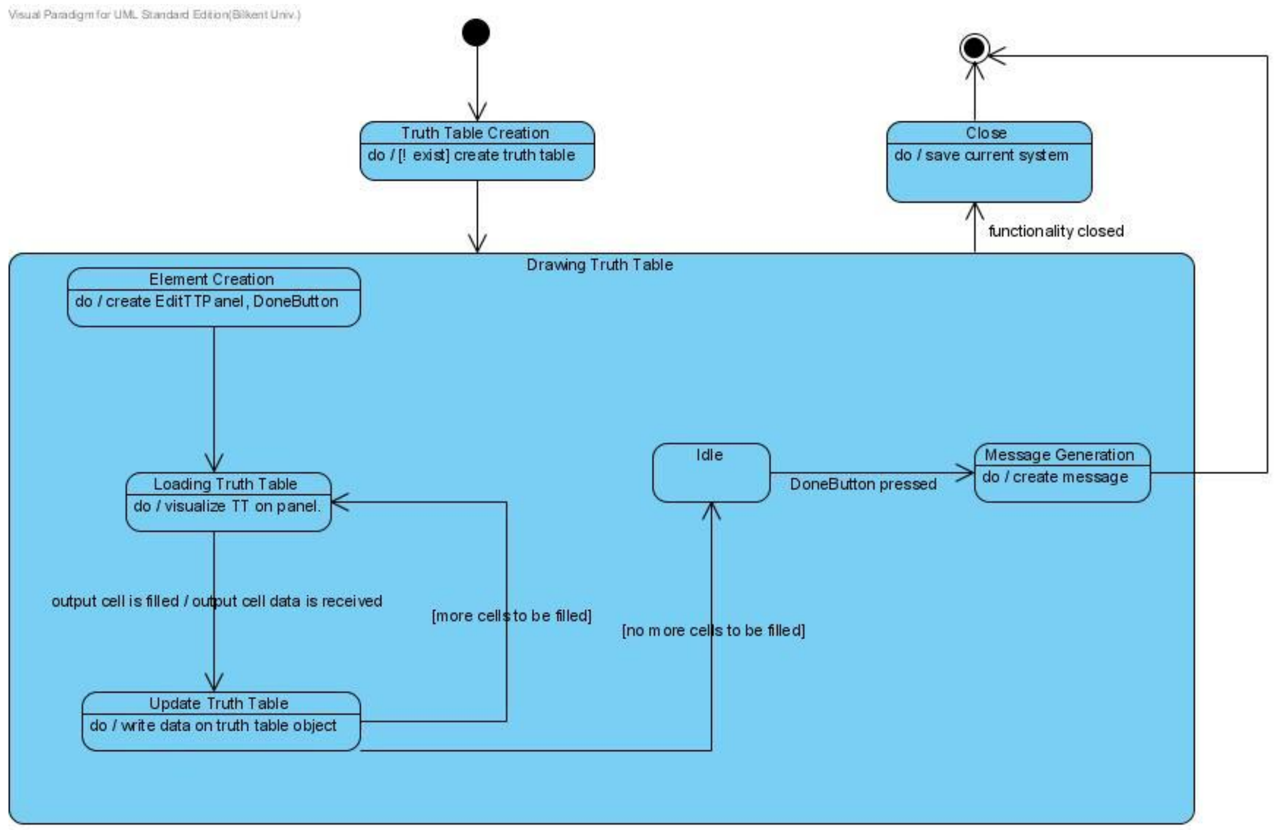

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Truth Table Creation
do / [! exist] create truth table
Close
do / save current system
functionality closed
Drawing Truth Table
Element Creation
do / create EditTTPanel, DoneButton
Idle
DoneButton pressed
Message Generation
do / create message
Loading Truth Table
do / visualize TT on panel.
output cell is filled / output cell data is received
[more cells to be filled]
[no more cells to be filled]
Update Truth Table
do / write data on truth table object

Figure 34 - State Chart for "ManageDrawTTControl"

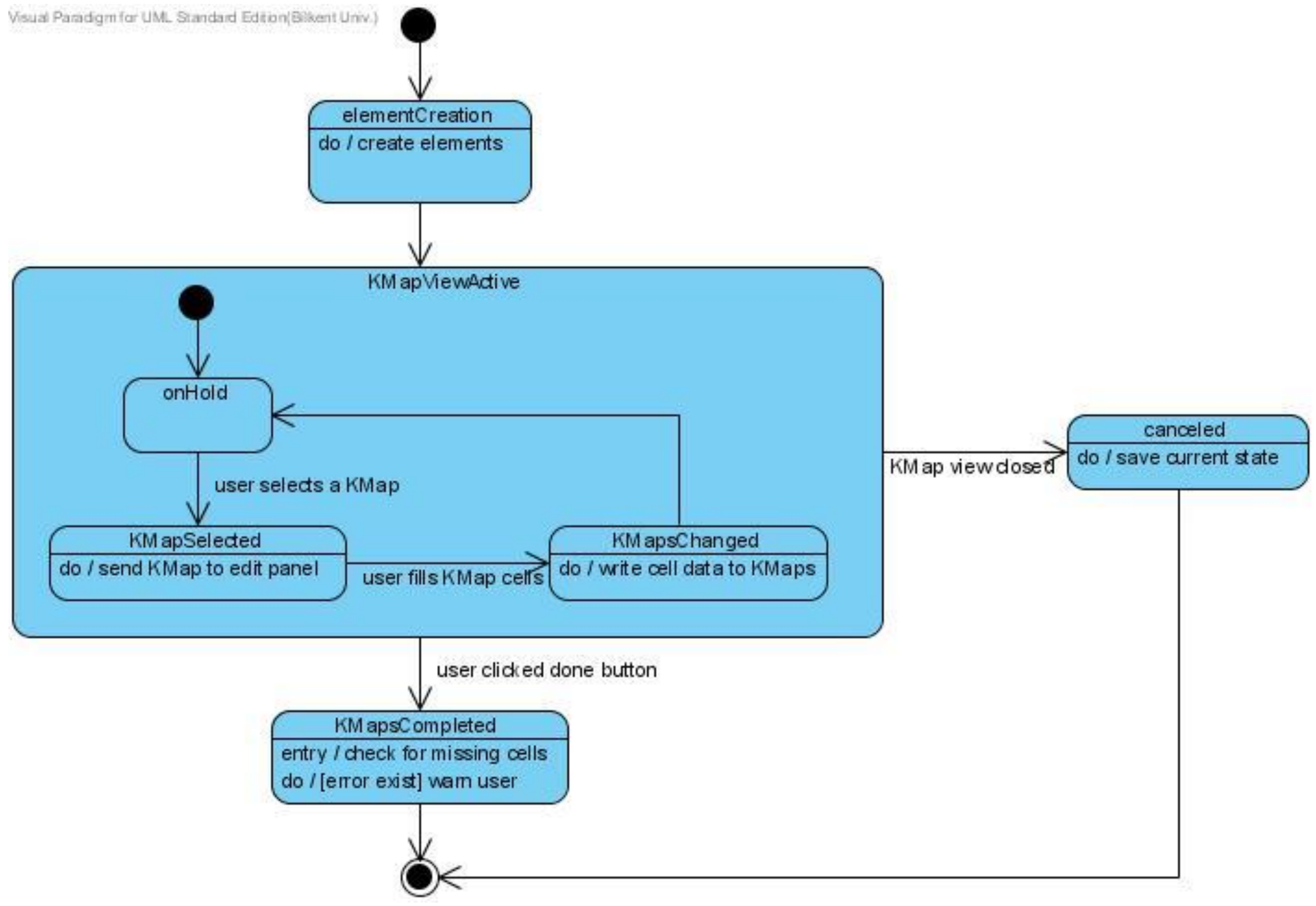


Figure 35 - State Chart for "ManageDrawKMapControl"

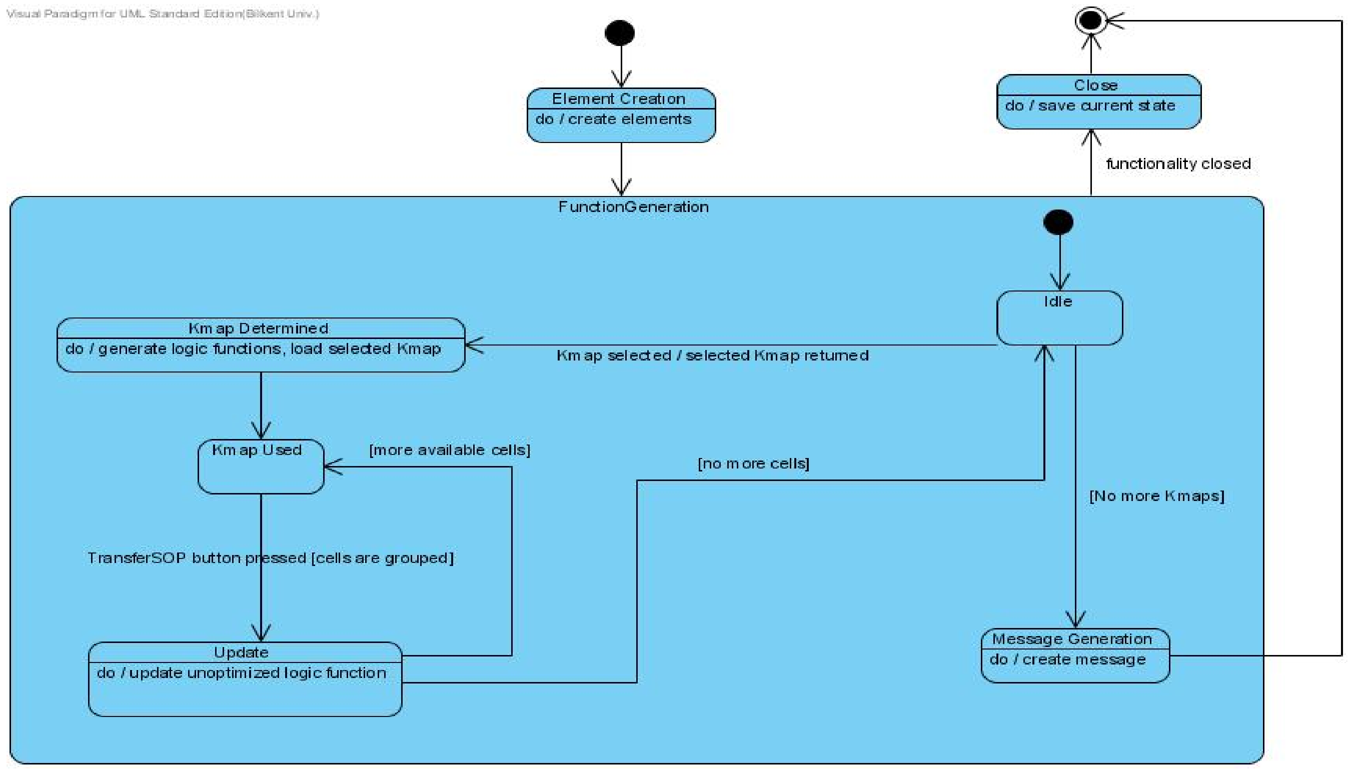

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Element Creation
do / create elements
Close
do / save current state
functionality closed
FunctionGeneration
Idle
Kmap Determined
do / generate logic functions, load selected Kmap
Kmap selected / selected Kmap returned
Kmap Used
[more available cells]
[no more cells]
[No more Kmaps]
TransferSOP button pressed [cells are grouped]
Update
do / update unoptimized logic function
Message Generation
do / create message

Figure 36 - State Chart for "GenerateFunctionControl"

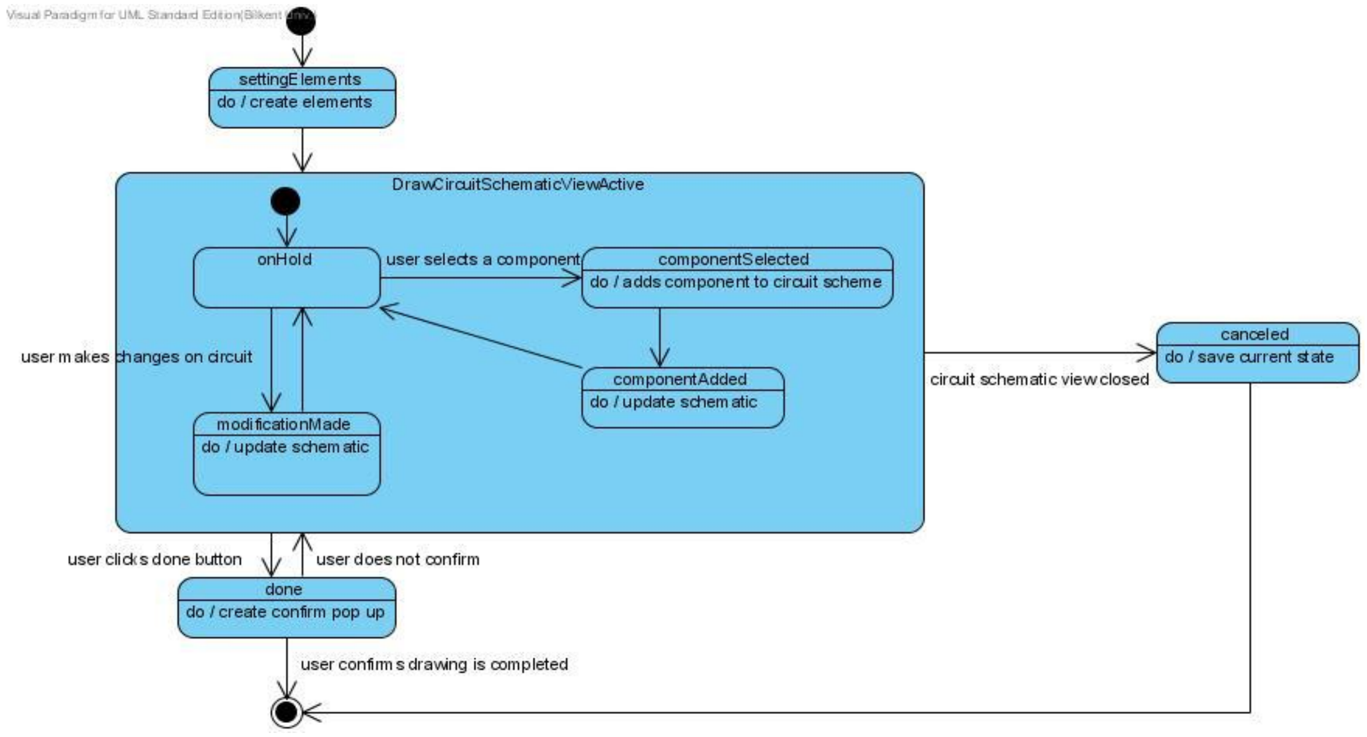

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
settingElements
do / create elements
DrawCircuitSchematicViewActive
onHold
user selects a component
componentSelected
do / adds component to circuit scheme
user makes changes on circuit
componentAdded
do / update schematic
modificationMade
do / update schematic
canceled
do / save current state
circuit schematic view closed
user clicks done button
user does not confirm
done
do / create confirm pop up
user confirms drawing is completed

Figure 37 - State Chart for "DrawCircuitSchematicControl"

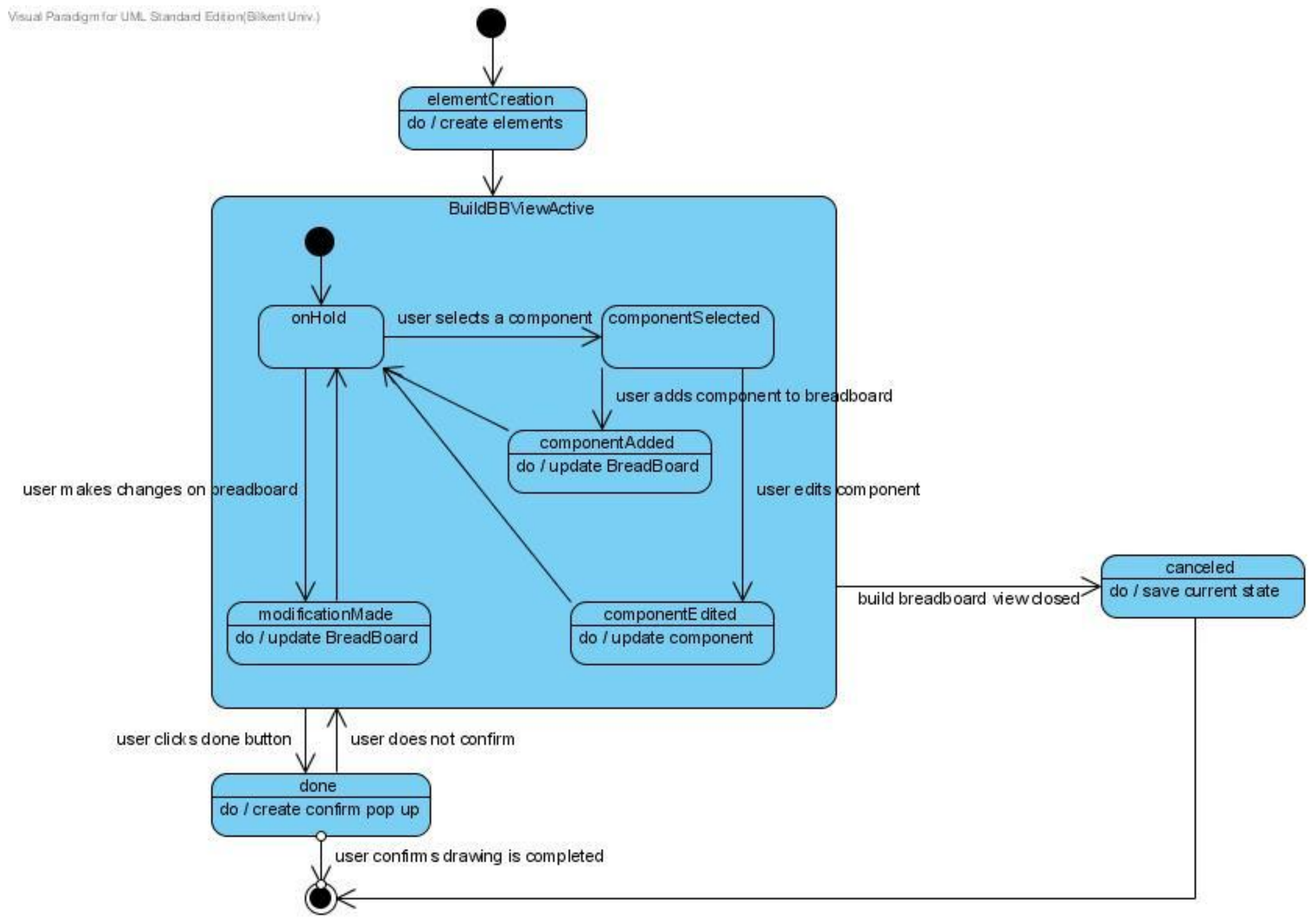
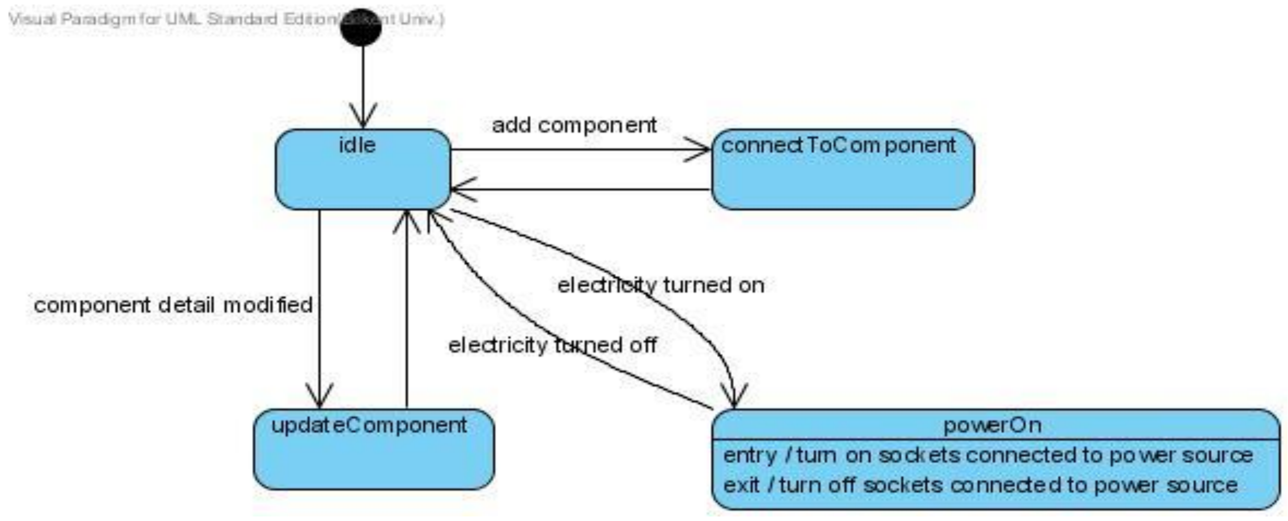


Figure 38 - State Chart for "BuildBBControl"



Visual Paradigm for UML Standard Edition(Bilkent Univ.)
idle
add component
connectToComponent
electricity turned on
electricity turned off
component detail modified
updateComponent
powerOn
entry / turn on sockets connected to power source
exit / turn off sockets connected to power source

Figure 39 - State Chart for "Breadboard"

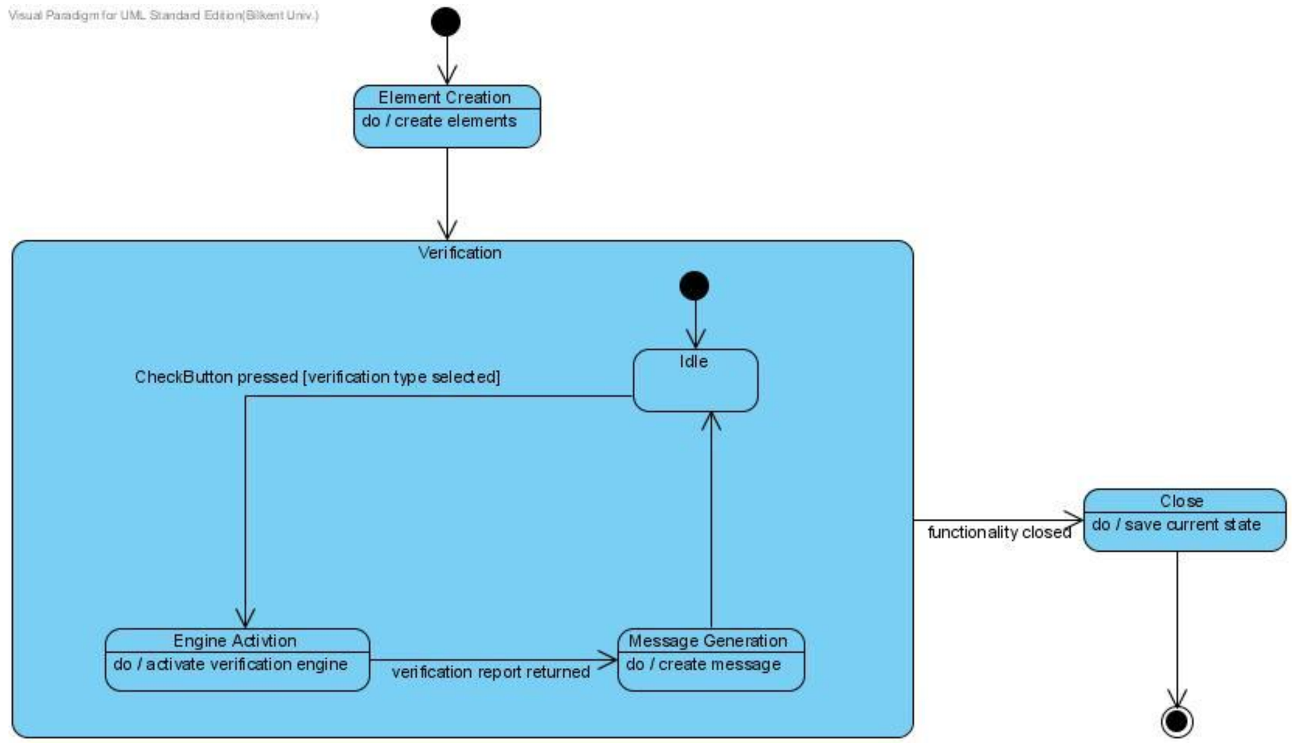

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Element Creation
do / create elements
Verification
Idle
CheckButton pressed [verification type selected]
Engine Activtion
do / activate verification engine
verification report returned
Message Generation
do / create message
functionality closed
Close
do / save current state

Figure 40 - State Chart for "VerifyControl"

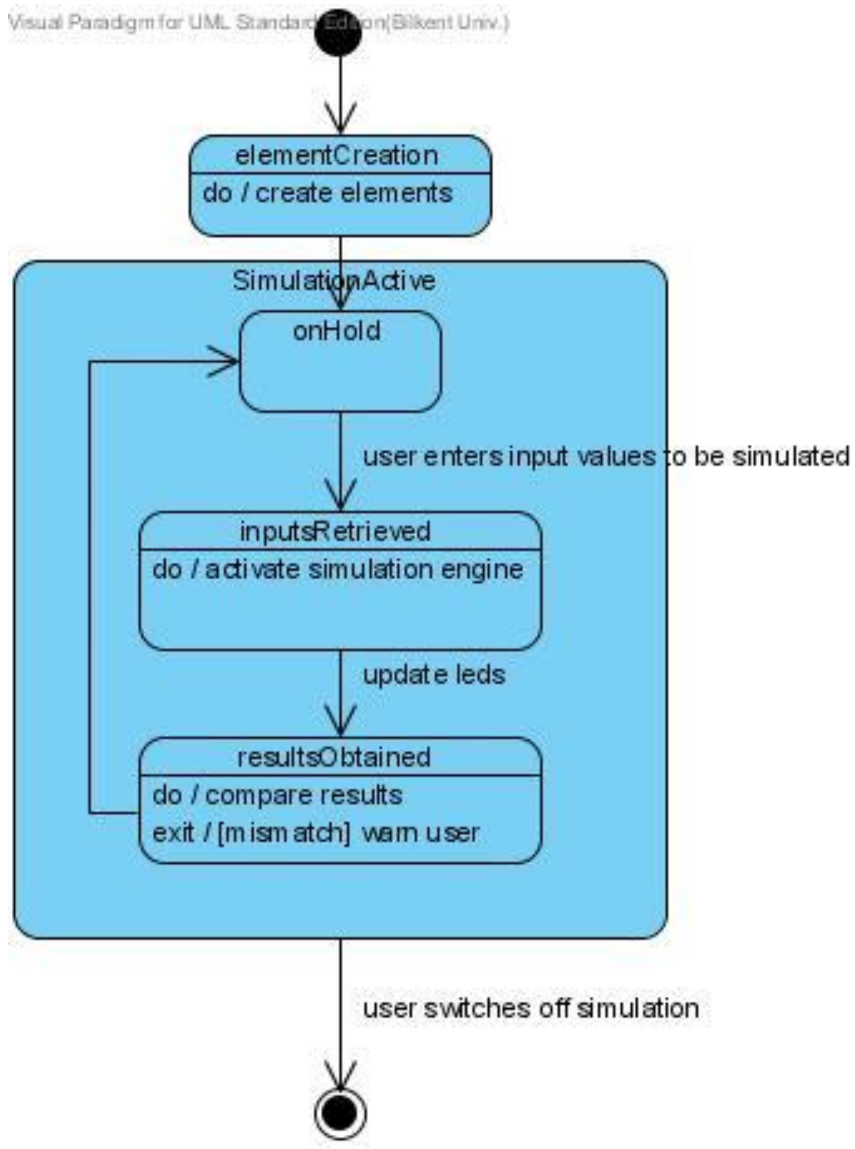
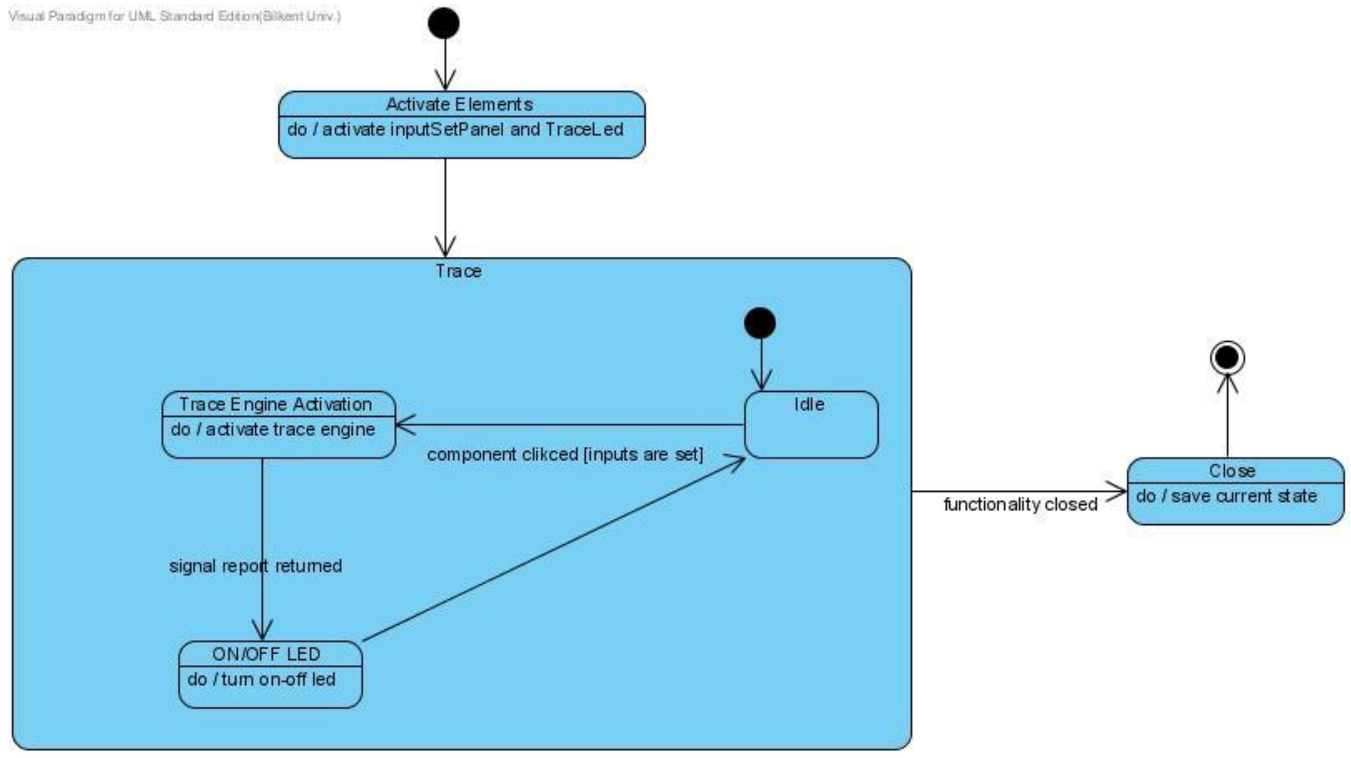

Visual Paradigm for UML Standard Edition(Bilkent Univ.)
elementCreation
do / create elements
SimulationActive
onHold
user enters input values to be simulated
inputsRetrieved
do / activate simulation engine
update leds
resultsObtained
do / compare results
exit / [mismatch] warn user
user switches off simulation

Figure 41 - State Chart for "SimulationControl"



Visual Paradigm for UML Standard Edition(Bilkent Univ.)
Activate Elements
do / activate inputSetPanel and TraceLed
Trace
Trace Engine Activation
do / activate trace engine
Idle
component clikced [inputs are set]
signal report returned
ON/OFF LED
do / turn on-off led
functionality closed
Close
do / save current state

Figure 42 - State Chart for "TraceCircuitControl"

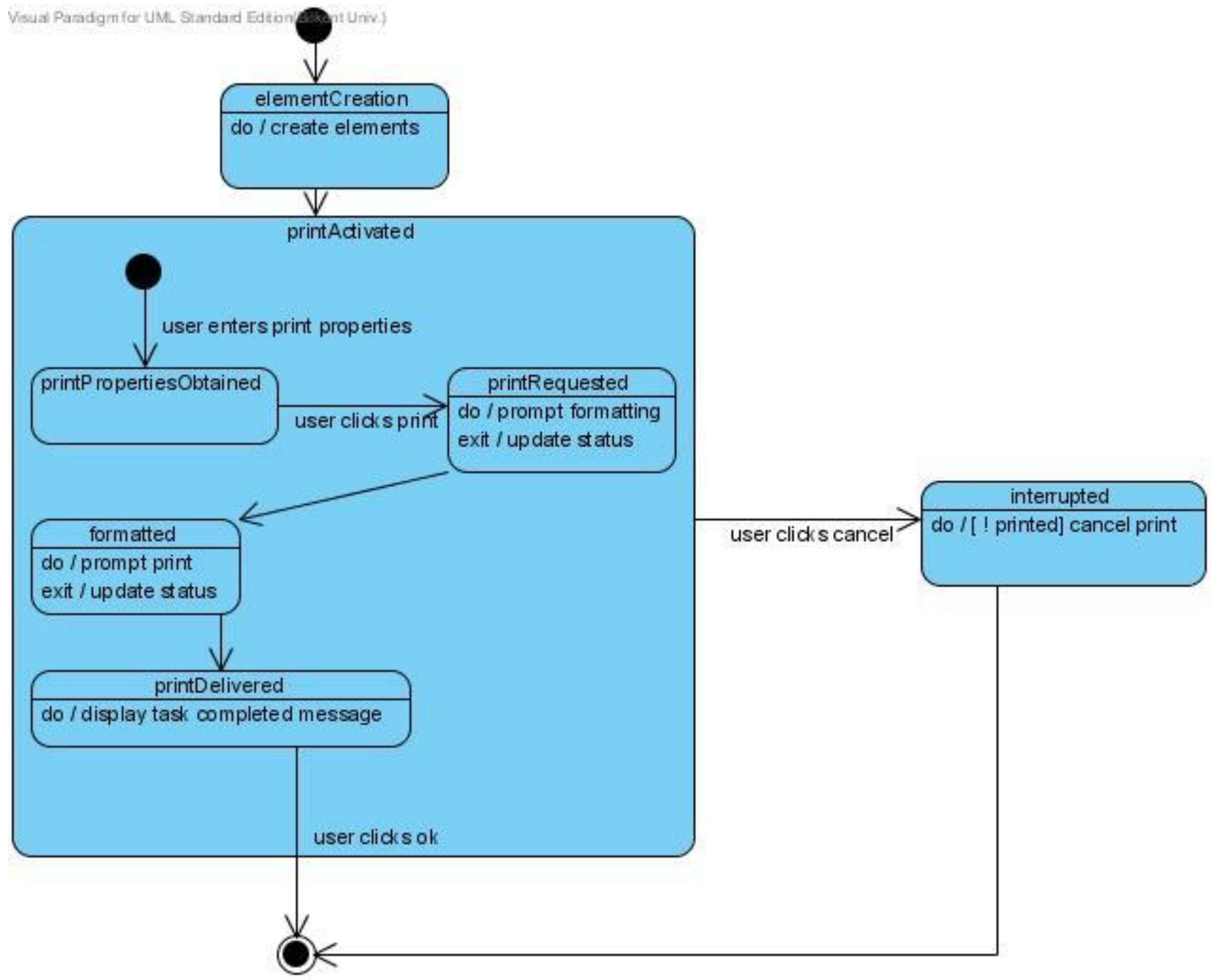


Figure 43 - State Chart for "PrintControl"

5. SYSTEM DESIGN

5.1. Purpose of the System

The aim of our project Logic Design and Breadboard Simulator is to provide a virtual environment to design and build logic circuits on breadboard. Designing and building a logic circuit include various processes. These processes could be described as circuit design, drawing circuit schematic, building circuit on breadboard, simulating circuit and verifying correctness and completeness of the circuit. We intended to include all of these processes in our project. In fact, all these steps are related to each other. To exemplify, the data which is created in circuit design may possibly be needed to draw circuit schematic. Therefore, program will provide interactions between different phases. Additionally, a new project must be created for each single circuit which user is intended to work on. Each project could be saved and loaded back.

5.2. Design Goals

Our project, Logic Design and Breadboard Simulator, will be designed as a single user application.

5.2.1. Ease of Use

The main goal of the system is to provide a user-friendly and plain interface to the user. Since the system is slightly complicated, we intended to ease the usage of the program. Aim of any user interface component should be clear to prevent confusions.

Student should be directed to follow the subsequent successive steps while designing and building his circuit.

- Enter the number of inputs and outputs together with their names.
- Draw Truth Table.
- Draw Karnaugh Map.
- Generate Logic Functions.
- Design Circuit.
- Build Circuit on Breadboard.
- Simulate Circuit.
- Verify correctness of system.

However, following this path should not be obligatory. Student should be able to start designing and building his project at any independent step. For example, it is possible to start designing the circuit before drawing the truth table. However, if he chooses to design in this way, he will not be allowed to verify his circuit using Truth Table because they are dependent on each other.

5.2.2. Reliability

The program will provide automated saving feature to store the work of the user at the predefined directory of the project. If System is interrupted at any step because of a crash, Student should be able to load the project without any data loss later.

5.2.3. Robustness

The program should be able to handle the invalid user inputs. If the user enters an invalid or unexpected input, system should warn the user about the condition immediately. Then, it should return to the latest stable state of the program that does not contain any of these invalid inputs.

5.2.4. Portability

The program should run on both Linux and Windows machines. This suggests that the program should ensure ease of portability between different environments. Moreover, the projects that are created by the user should be saved into a directory of files. Then it should be possible to use this directory of files in some other copy of the program. It should be allowed to view, change, or edit the project independent of the program copy used on some other platforms.

5.2.5. Maintainability

The source code should be written in a generic way that will make it easier to understand and contribute the coding process while implementing the project. It is expected to generate a common way of implementation and using this throughout.

5.2.6. Documentation for Open Source Development

The documentation of the program should be well-organized to make it easier for new developers to understand the system. The source code documentation, analysis and design reports should be readable and understandable. By that way, developers could contribute new features to the system easily.

5.3. Subsystem Decomposition

We have firstly decomposed the system into smaller parts based on the use case and analysis models. After that, we have applied the Repository and Three Tier architectural patterns to these subsystems.

5.3.1. Initial Subsystem Decomposition

We have derived the initial subsystem decomposition from the functional requirements and the use case models. Starting from the very beginning of our project design – use case modeling – we have kept the functionally related objects together. We simply assigned the participating object packages that we have identified in each use case to the subsystems. Then, being aware of the storage requirement, we have created a dedicated subsystem for objects used for moving data among subsystems and named it as XMLStorage. At the end, to minimize the number of associations – reducing coupling – crossing subsystem boundaries, we have introduced the BBSimParser subsystem. The details for these subsystems are explained in the following subsection, 5.3.2.

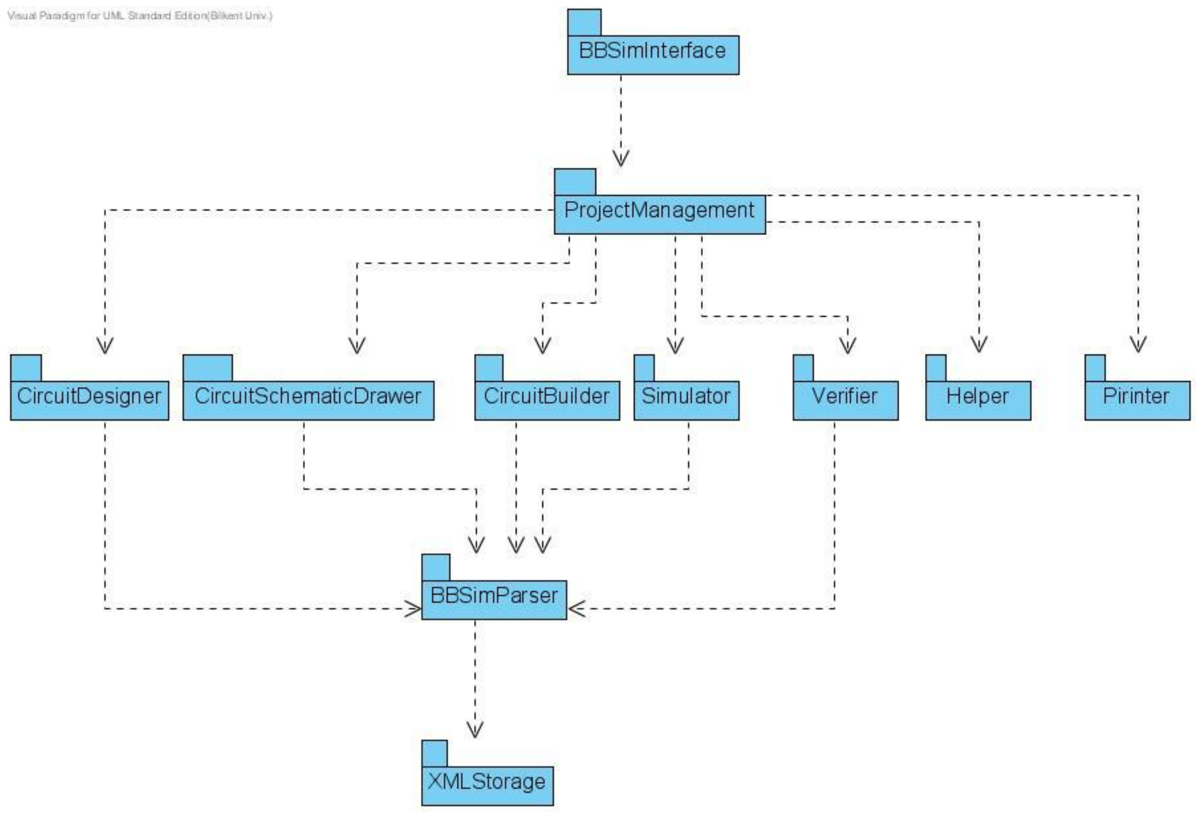


Figure 44: Initial Subsystem Decomposition for BBSim Project

5.3.2. Detailed Subsystem Decomposition

After designing the initial subsystem decomposition, we have started to detail the subsystems that we have introduced. We basically have listed the subsystems and then given the classes involved in these systems. We also tried to keep high cohesion within the subsystems.

5.3.2.1. BBSimInterface

BBSimInterface consist of MainFrame and Transition classes as it is seen in Figure 45. MainFrame class handles the view of the user and is updated with the invocations coming from the ProjectControl class of the ProjectManagement subsystem. Transition class handles the transitions from one view to the other. If there are things left to do in one view, it will not allow the user to change the view of the program.

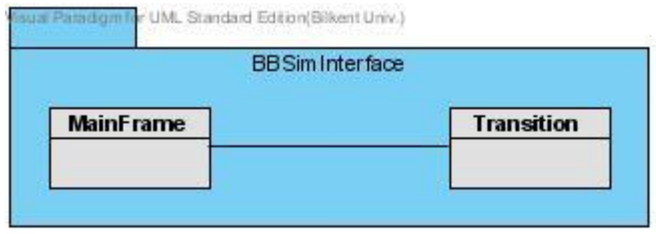


Figure 45: BBSimInterface Subsytem

5.3.2.2. ProjectManagement

ProjectManagement consist of Project, ProjectControl, CreateProjectControl, LoadProjectControl and SaveProjectControl classes as it is seen in Figure 46. Project class is being created whenever the user wants to design a new project using our program. The whole process of designing is stored in this class's attributes. It will communicate with the control objects within the ProjectManagement subsystem and coordinates the whole work of the user. ProjectControl class controls the flow of the design process. It will warn the BBSimInterface subsystem to change the view of the user according to the required fields within the design. CreateProjectControl class handles the project creation process within our program. LoadProjectControl class handles the project loading process within our program. When the user wants to load a project that he has developed earlier, an object of this class is created. SaveProjectControl class handles the project saving process within our program. When the user wants to save a project that he has been developing, an object of this class is created.

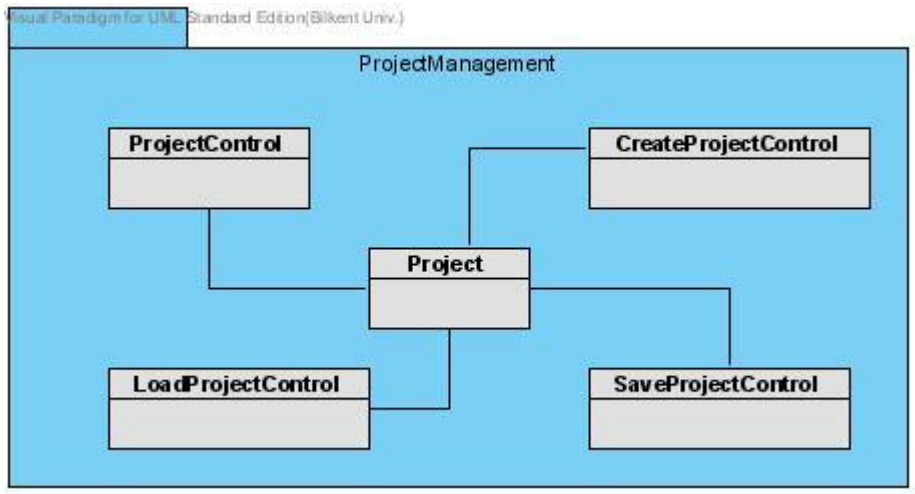


Figure 46: ProjectManagement Subsystem

5.3.2.3. DesignCircuit

DesignCircuit subsystem includes DrawTruthTable, DrawKarnaughMap and GenerateLogicFunctions subsystems. GenerateLogicFunctions subsystem and DrawKarnaugMap subsystems are connected to each other because the user will use the Karnaugh Maps that he has drawn to generate the logic functions. Also, ManageDrawTTControl, ManageDrawKarnaugMapControl and GenerateLogicFunctionsControl classes of these subsystems are connected to the ProjectControl class of the ProjectManagement subsystem.

DrawTruthTable subsystem includes ManageDrawTTControl, TTable, and EditTTPanel classes. ManageDrawTTControl class controls the Truth Table drawing process. It will inform the ProjectControl to show the EditTTPanel to the user. EditTTPanel gathers the content of the object created using TTable class and use it to build itself.

DrawKarnaughMap subsystem includes ManageDrawKarnaugMapControl, KMapListDisplay, EditKMapPanel, and KMap classes. ManageDrawKarnaugMapControl class controls the Karnaugh Map drawing process. It will inform the ProjectControl to show the EditKMapPanel to the user. EditKMapPanel gathers the content of the object created using KMap class and use it to build itself. It will also use the KMapListDisplay class because it is possible to have more than one KMap in a project.

GenerateLogicFunctions subsystem includes GenerateLogicFunctionsControl, GenerateFunctionsPanel, KMapListDisplay, DisplayFunctionPanel, OptimizedLogicFunctions and UnoptimizedLogicFunctions classes. GenerateLogicFunctionsControl class controls the Generating Logic functions process. It will inform the ProjectControl to show the GenerateFunctionsPanel and

DisplayFunctionPanel using the KMapListDisplay. It also gathers information from the OptimizedLogicFunctions and UnoptimizedLogicFunctions.

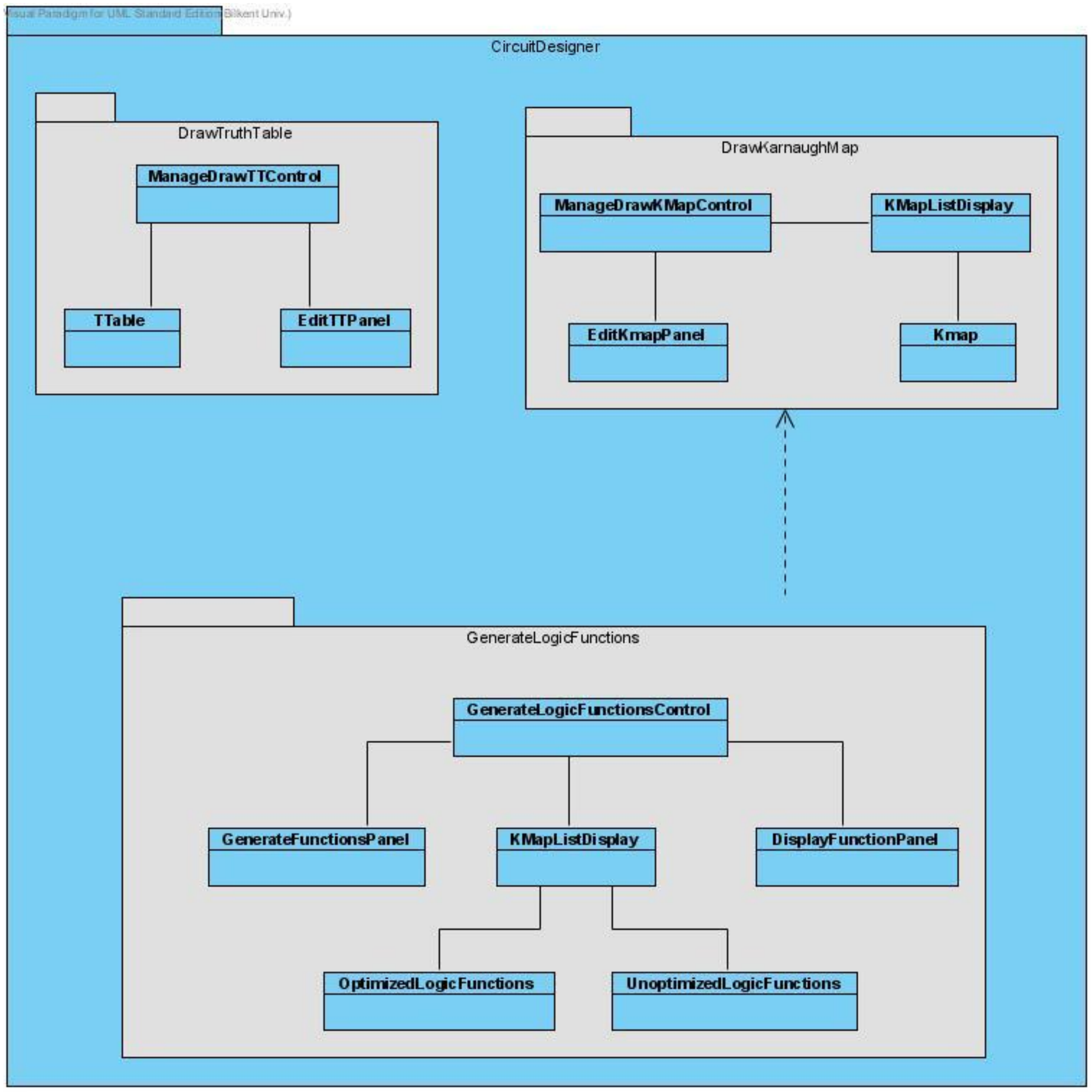


Figure 47: CircuitDesigner Subsystem

5.3.2.4. DrawCircuitSchematic

DrawCircuitSchematicControl is connected to the ProjectControl class of the ProjectManagement subsystem. It notifies the ProjectControl to display the LogicFunctionDisplay, ComponentEditPanel and ComponentListPanel to the user. CircuitSchematic includes the IO, Wire and Gate components within the project.

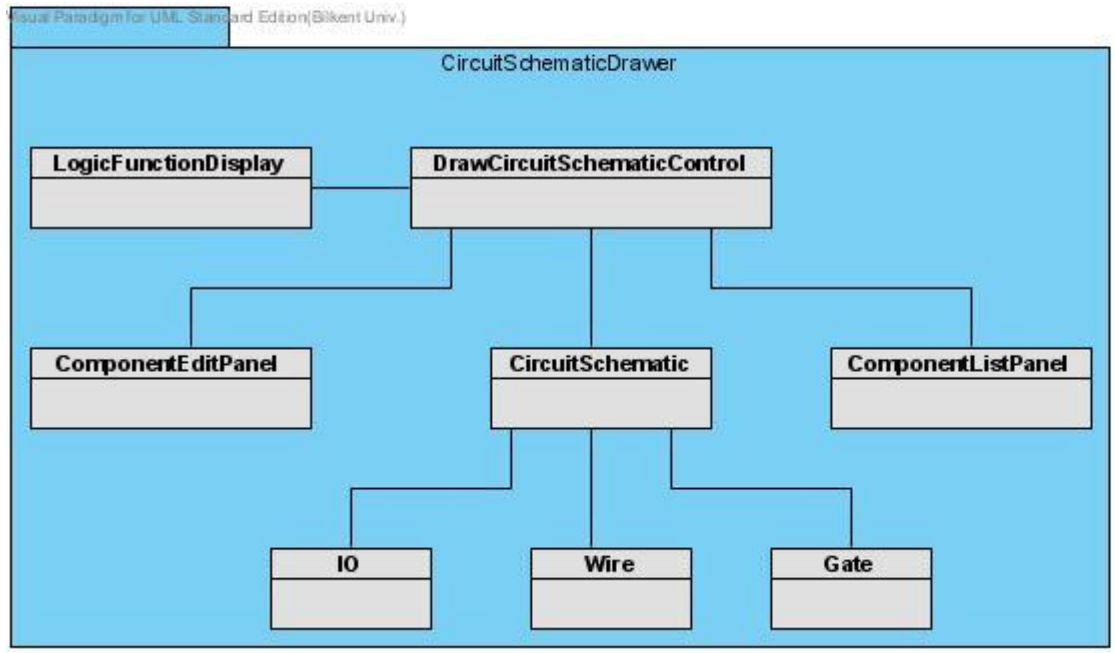


Figure 48: CircuitSchematicDrawer Subsystem

5.3.2.5. BuildCircuit

BuildCircuit is connected to the ProjectControl class of the ProjectManagement subsystem. It notifies the ProjectControl to display the BBEditPanel and ComponentListPanel to the user through BBSimInterface subsystem. There are Workstation and Breadboard classes that are connected to the control class. Workstation class has WorkstationSocket and Led classes to handle its job. Breadboard class has Chip, Wire and Breadboard components that are required to have functionality within the context.

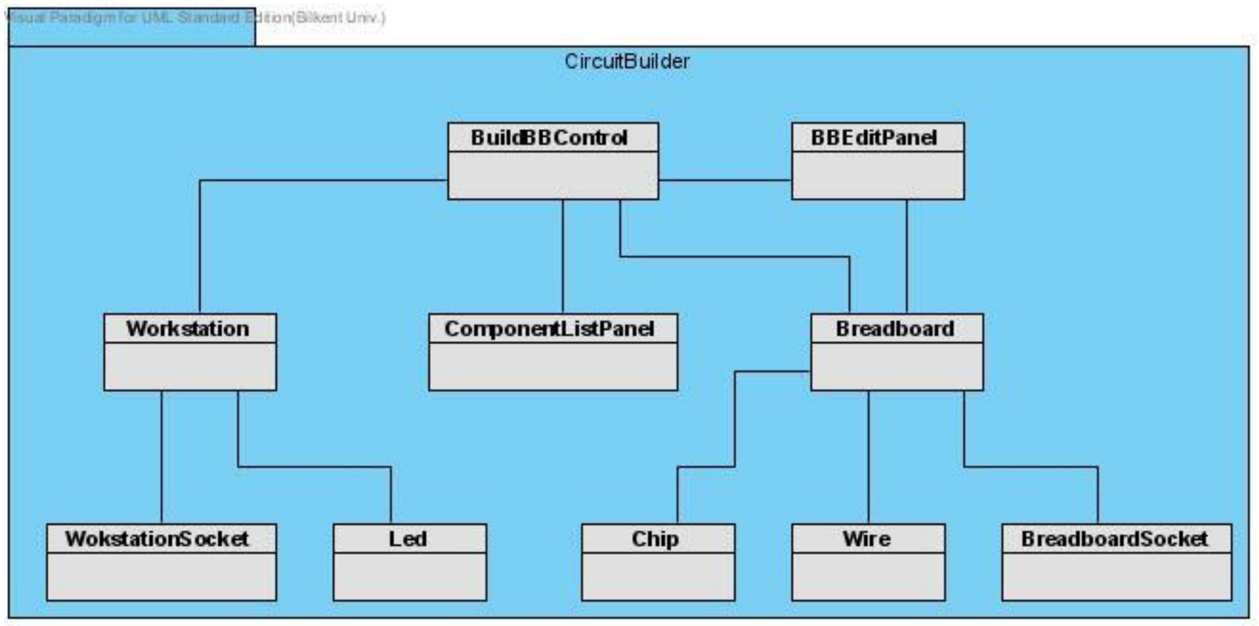


Figure 49: CircuitBuilder Subsystem

5.3.2.6. Simulate

Simulate subsystem includes the TraceCircuit subsystem. Its SimulationControl class notifies the ProjectControl to display the BBViewPanel to the user. There are Workstation and BBOutputTable classes that are connected to the control class. SimulationEngine class handles the process of simulating and informs the state of the simulation to the SimulationControl class. TraceCircuitControl is also using the SimulationEngine to handle the Tracing process with InputSetpPanel Interface and TraceLed.

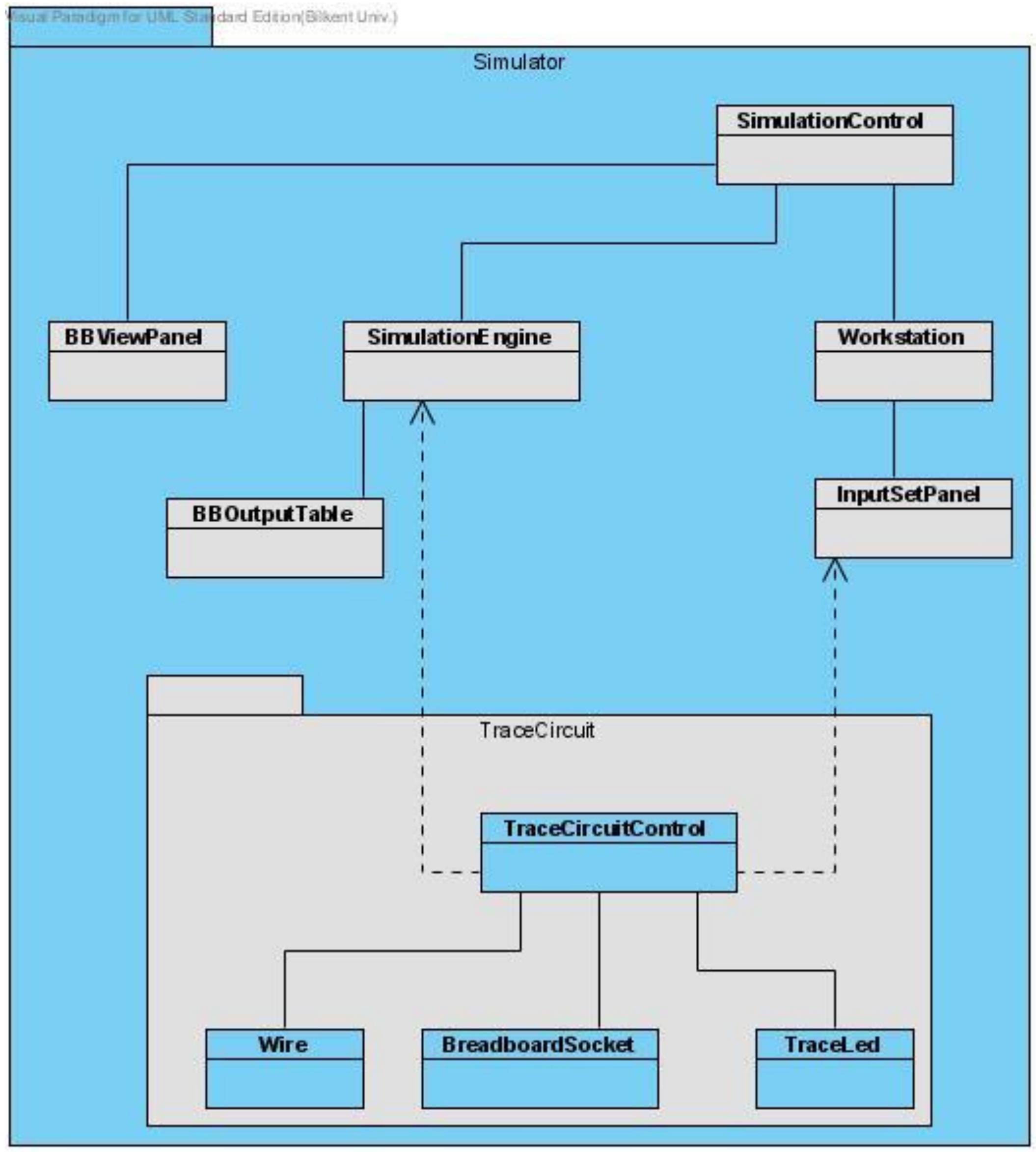


Figure 50: Simulator Subsystem

5.3.2.7. Verify

VerifyControl is connected to the ProjectControl class of the ProjectManagement subsystem. It notifies the ProjectControl to display the ResultWindow and StepSelectorList to the user through BBSimInterface subsystem. The process of Verification is handled by the VerificationEngine and is notified to the VerifyControl class.

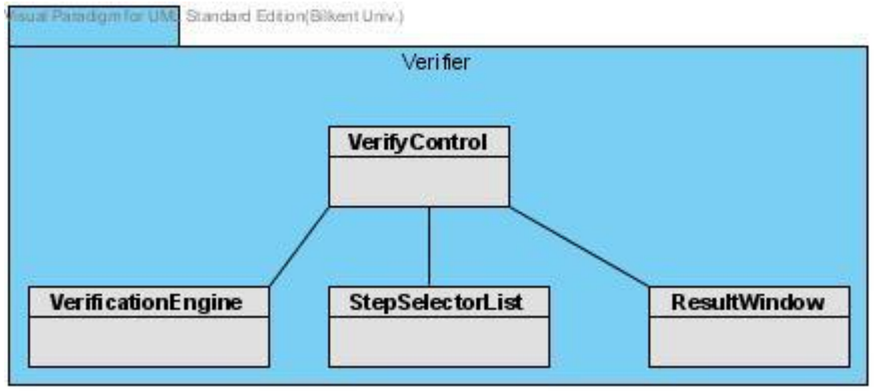


Figure 51: Verifier Subsystem

5.3.2.8. GetHelp

GetHelpControl is connected to the ProjectControl class of the ProjectManagement subsystem. It notifies the ProjectControl to display the HelpDisplayPanel to the user through BBSimInterface subsystem. The data that will be displayed to the user is fetched using the HelpDataFetcher class.

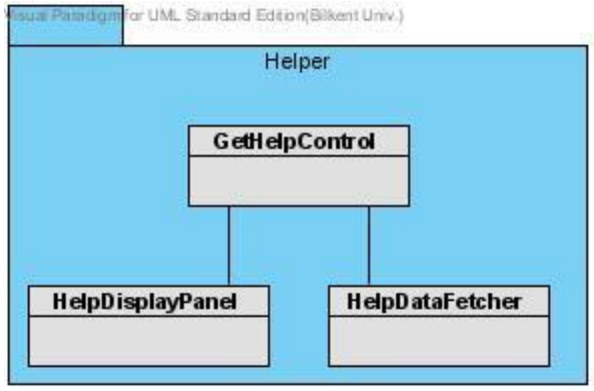


Figure 52: Helper Subsystem

5.3.2.9. Print

ProntControl is connected to the ProjectControl class of the ProjectManagement subsystem. It notifies the ProjectControl to display the PrintSettingsPopUp to the user through BBSimInterface subsystem. This interface will show the options that are listed in the PrintProperties class. Based on the selections of the users, PrintFormatter will shape the page that is requested to print and send it to the PrintControl. Also, the status of the Print job is held within the PrintStatus class.

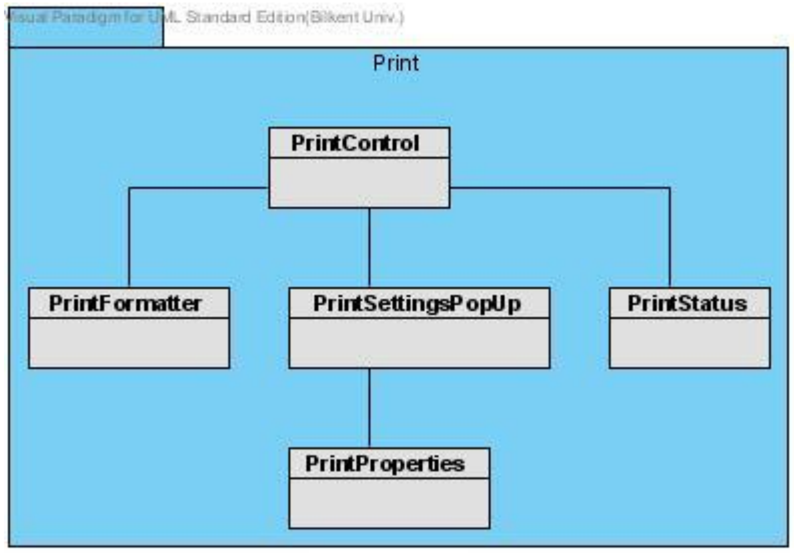


Figure 53: Printer Subsystem

5.3.2.10. BBSimParser

BBSimParser subsystem will fetch the data from the XMLStorage subsystem and send it to where it is requested. We have separated the fetching and storing processes into two different subsystems to have lower coupling.

5.3.2.11. XMLStorage

XMLStorage consists of different XML files storing the projects that are developed by the user. We have come up with this idea based on our design goals. To make it easier for changes and reduce coupling, we have selected to divide the Storage and Parser into two different subsystems.

5.4. Architectural Patterns

After decomposing the system, we have analyzed the subsystems and decided to apply Repository and Three-tier patterns. The reason for choosing these patterns, their structure, and the subsytems involved in them are explained in the following sections.

5.4.1. Repository Pattern

Our central repository for storing data is XMLStorage. Other subsystems are accessing and modifying the data on XMLStorage using the services provided by the BBSimParser subsystem. The reason for separating these services is to achieve the purpose of having low coupled system. The central location of the data makes it easier to deal with concurrency and integrity issues between subsystems.

We have chosen this type of architecture because our data that is created by the users of our program is going to be constantly changing. These constant changes require complex data processing tasks and also these tasks should keep the program consistent.

We have included DesignCircuit, DrawCircuitSchematic, BuildCircuit, Simulate and Verify subsystems of our project in this pattern. Since the users of our program will create a project and base their design on these subsystems, we have only selected the subsystems that are constantly changing to store in the XML files. Each of these subsystems will be stored in a different file. However the structure of these files will be almost similar.

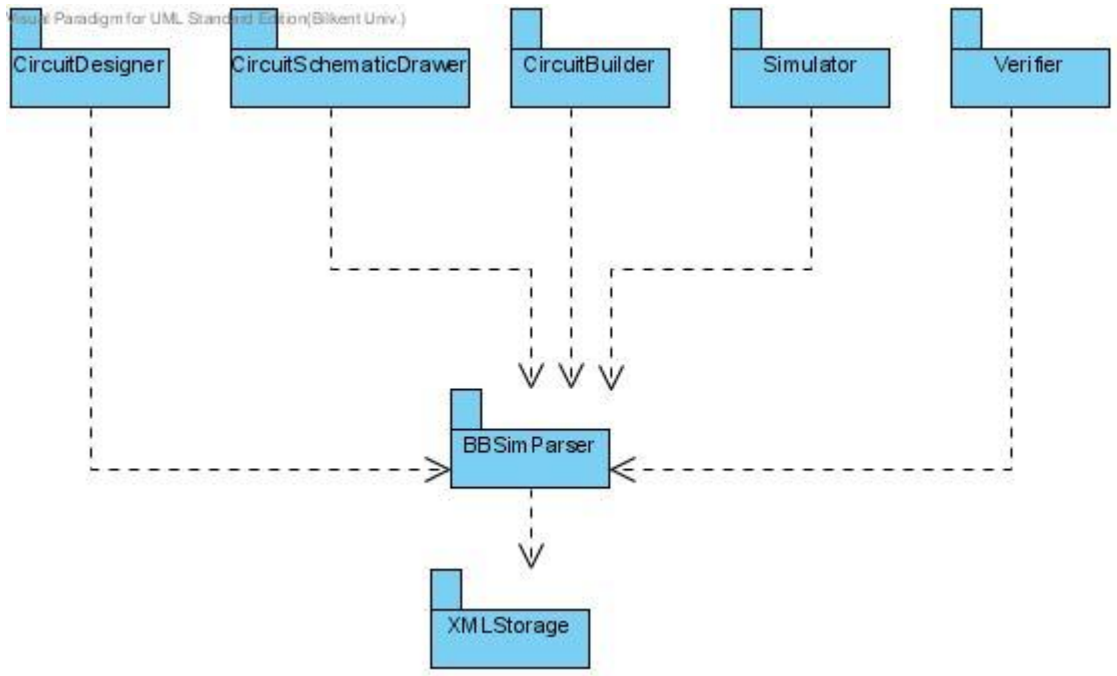


Figure 54: Repository Pattern Architecture

5.4.2. Three Tier Pattern

In our project, we decided to implement interface, internal application of system and storage separately. We aimed to separate concerns as much as possible. By that way, we tried to make whole system more resilient to changes. To exemplify, changes on interface should not affect implementation of internal application. Since we divided whole system into three main parts, we decided to use "Three-tier" architectural pattern.

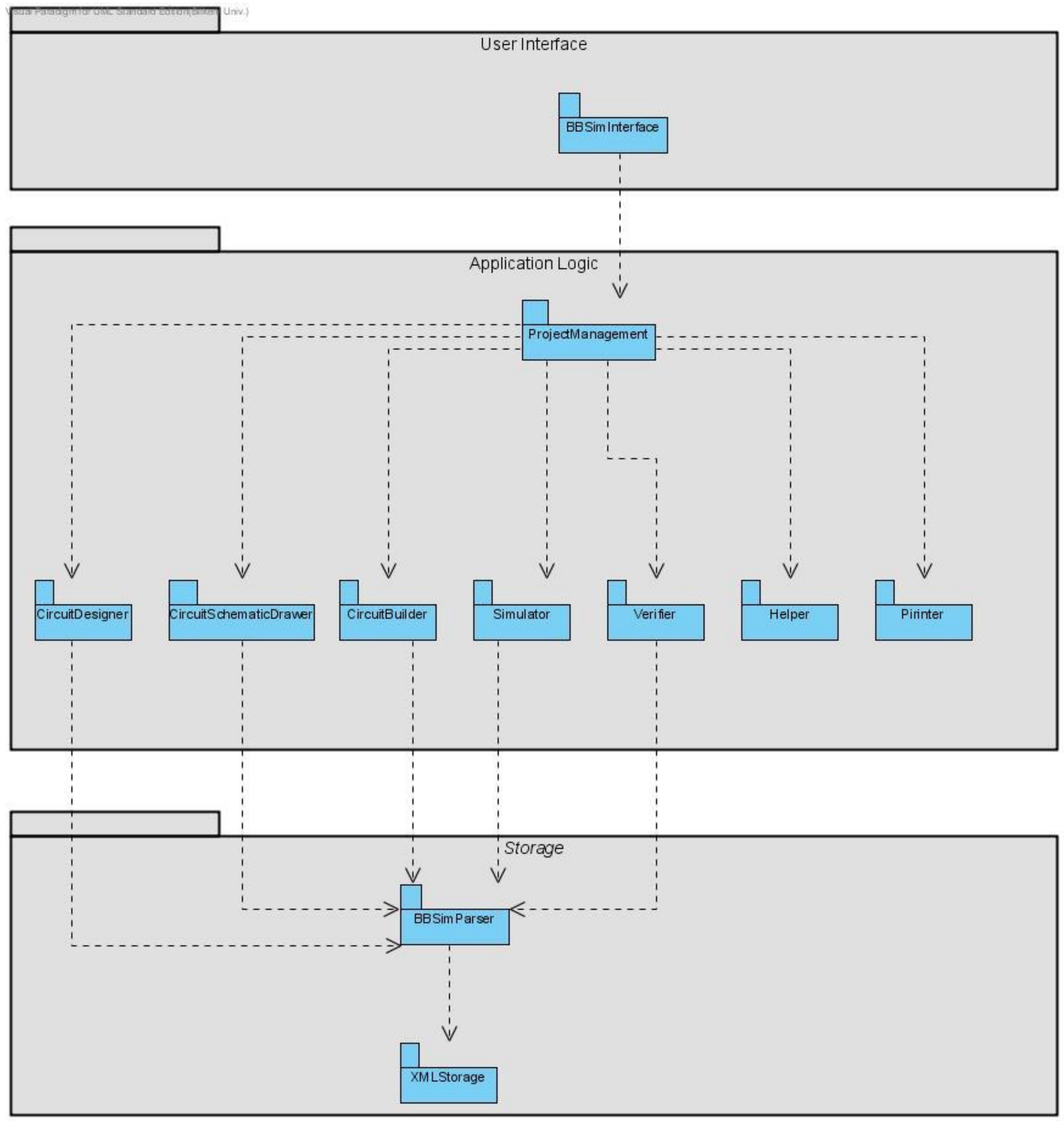


Figure 55: Three Tier Pattern Architecture

5.5. Hardware/Software Mapping

We have decided to use a WebServer to store the Help data of our project. When the users want to get help about our program, they will be directed to a web server using their default web browser. Since our design goals include operation on Windows and Linux machines, we have shown examples of both in the Figure 56.

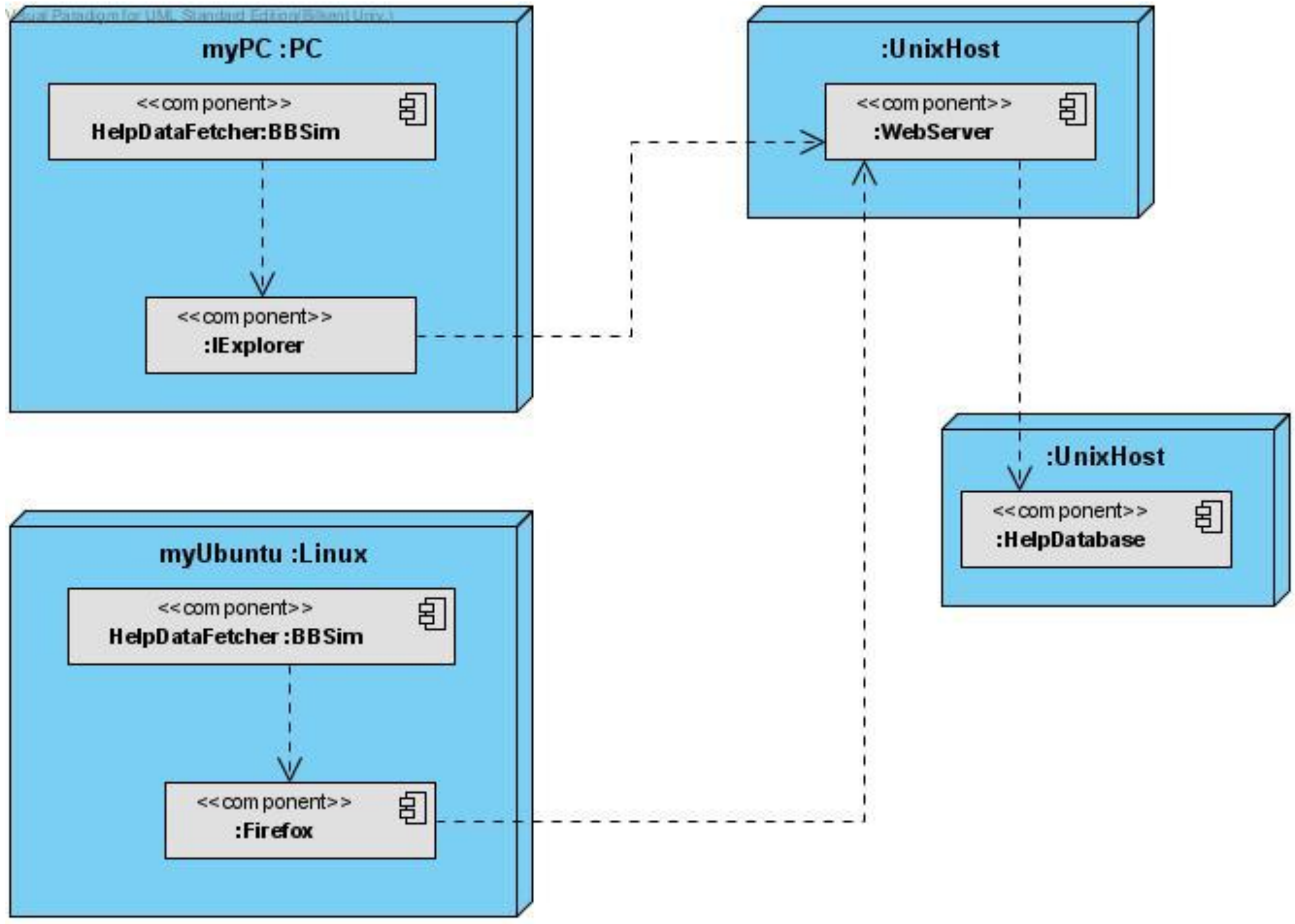


Figure 56: Allocation of Help subsystem to hardware

We have also decided to make it possible for our users to print the work that they have done at any time they desire. So, we have deployed a Printer as illustrated in Figure 57 below. When the users of our program want to print some of its parts, the project will consider formatting the paper and sending the data from our program to the computer through the PrinterControl class.

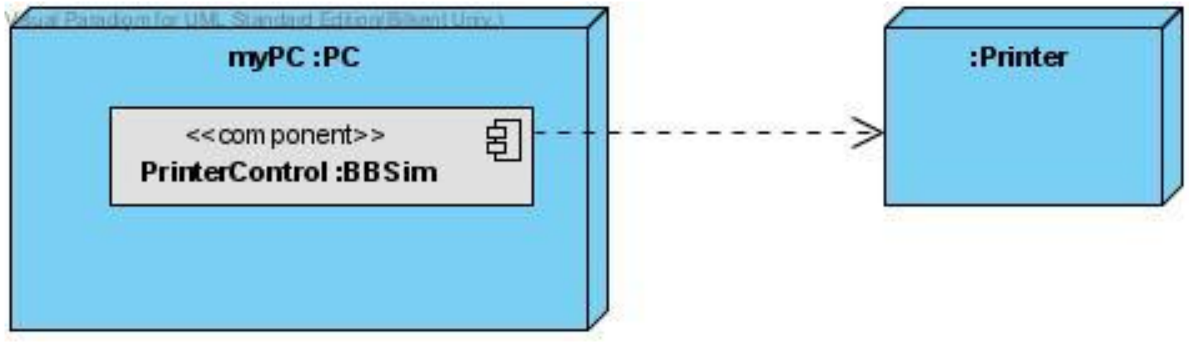


Figure 57: Allocation of Print subsystem to hardware

5.6. Persistent Data Management

In the following subsections, we have identified the persistent objects that should be stored in a repository environment and explained the storage management strategy that we have selected to store these persistent data objects.

5.6.1. Identifying Persistent Objects

While designing our project, we have realized that it is necessary to have a repository environment that stores 2 things. The first one is the projects that will be developed by the users of our program, BBSim. Since the content of these projects will be persistent, we need to store them at some place to enable the users to work on them any time they desire. And the second one is the data that will be used in the Help feature of our program. Depending on the needs of the users, we will need to update the data that will be displayed in the Help feature of our program. This requirement suggests us to use a persistent data object.

5.6.2. Selecting a Storage Management Strategy

We have decided to use different storage management strategies for the persistent data objects that we have identified before. Both of them are explained in the following two paragraphs.

Our portability design goal entails that the projects that will be developed by the users of program should be transportable to another copy of BBSim program. To provide users for transferring their projects from one platform to another, we first decided to use flat files for storage. Such a system can be installed easily, since there is no need to have a database management system to configure or to manage. However, a system based on just flat files would not scale to the projects that will be developed by the users. Thus, analyzing the existing file management systems, we have decided to use XML files to store the projects developed by the users. The reason for choosing XML files is that they allow us to define an independent file structure with independent tags and use them throughout the project.

To store the help data, we need to have some storage system that is not on the machines of our users. This requirement has driven us to use a Web Server that stores the contents of our Help feature. When users click on the Help button, we decided to direct them to this Web Server using the default Web Browser on the machines that they are using. However, this reminded us the possibility of not having an internet connection at any time. Analyzing the Help features of some existing programs (Eclipse, Netbeans etc.), we have also decided to store the latest updated version of the Help data on the local machines of the users. Whenever the users click on the Help button, the program will show them this local copy of the Help data. At the same time, if we detect an internet connection, program will automatically check the list of Help data on the Web Server and warn the users when there is any new Help data arrived.

5.7. Access Control and Security

5.7.1. Access Control

Our project *Logic Design and BreadBoard Simulator* is a system which consists two users. In fact, one of the users, *Student,* have access to all classes. However, the other user, *Assistant,* have a limited access on the system. To document access rights, we draw an access control matrix depicting the allowed operations on entity objects for each actor. For the representation of access control matrix, we choose global access table. The reason is that an access control list and a capability list handle the access issue in a limited manner. Access control lists make it faster to answer the question "Who has access to this object?", however, these lists are not practical to determine all objects which accessed by a specific actor. Conversely, a capability list makes it faster to answer the question "Which objects has this actor access to?", however, these lists are not practical to determine all actors which access a specific object. Differently, global access table handles access issue in terms of both objects and actors. It is easy to determine objects which accessed by a specific actor and also it is easy to determine actors which access to a specific object.

Student is a user which benefits from system to produce a whole digital design work. Together with that, *Assistant* interacts with system to check completeness and correctness of *Student*'s work.

To clarify, *Student* can create *Project, TruthTable, Kmap, OptimizedLogicFunction, UnOptimizedLogicFunction, CircuitSchematic, BreadBoard, Workstation, TraceLed,VerificationList* and *Help* objects. Additionally, *Student* have right to invoke some operations on these classes. Apartly, *Assistant* can create *WorkStation, VerificationList* and *Help* objects. Similarly, *Assistant* have right to invoke some operations on these three entity objects.

	Project	**TTable**	**KMap**	**Optimized Logic Function**	**Unoptimized Logic Function**	**Circuit Schmematic**
Student	Create Save Load	Create Fill Table View	Create Fill Map View	Create Func. Update Func.	Create Func. Update Func.	Set Wire Info Set Gate Info Set Input Set Output
Assistant	-	-	-	-	-	-

	Workstation	**TraceLed**	**Verification List**	**Help**
Student	Activate Set Led Connection Info Set Switch Value	Set Trace Led Connection View Led Status	Select Verification Type	Get Help Data
Assistant	Activate Set Led Connection Info Set Switch Value	Set Trace Led Connection View Led Status	Select Verification Type	Get Help Data

Figure 58: Access Matrix for BBSim System

5.7.2. Security

Our system have a feature that it can save and load projects. However, it is risky that some unrelated people could load a student's project and spoil it. To eliminate such an issue, we decided to add an authentication feature. By that feature, when a project is saved, user will assign a password for that project. At the load time, system will ask for the project's password. Any person who does not know password could not be able to load project and work on it. In fact, only *Student* is exposed to authentication process. Since *Assistant* only check work done, there is no need of authentication for him.

5.8. Global Software Control

Among three possible control flow mechanisms, which are *Procedure-driven control*, *Event-driven control* and *Threads,* we decided to use *Event-driven control flow* for our project.

At first, we did not choose *Procedure-driven control flow,* which is mostly used in systems written in procedural languages, because the flow is arranged according to sequence of inputs and it is hard to determine order of inputs for large number of objects. Therefore, we consider benefiting from *Procedure-driven control* in the testing of subsystems, not for whole system.

Secondly, we did not use *Threads* because they cause problems about debugging and testing. More mature tools are needed to develop software with *Threads*.

As mentioned, we decided on *Event-driven control flow*. It is the most mature mechanism. Additionally, it supports object oriented programming. In our system, each subsystem has a unique independent control object that is responsible for managing the events of the corresponding subsystem. By localizing control flow from whole system to subsystems, we aimed to make system more resilient to changes in control flow implementation. In our system, although control flow is localized to subsystems, it is centralized in subsystems by having unique control objects for each subsystem. It means only one control object manages a subsystem. Centralization on control of subsystems makes it easy to change control structure.

5.9. Boundary Conditions

Although we have dealt with designing and refining the system, we still need to examine the boundary conditions of the system – which are, to decide how the system is started and shut down – and we need to define how we deal with major failures such as data corruption.

5.9.1. Configuration

Most of configuration related conditions, which are creation and destruction of persistent objects are handled in revised versions of use cases. We have one remaining, which is listed and explained below:

Create Engines	As soon as the Student creates or loads a Project; the engines of the system are created (SimulationEngine, VerificationEngine and TraceEngine). They are configured in their own uses cases in later steps, and destroyed when the project is closed.

As for an exceptional configuration condition, we have help feature. Because it is on a Web Server and will be changing based on the users need, we will not make any contacts with the copies of our program used by others. To configure the data for Help, we will just need to have an SSH[1] enabled website. Using SSH, developers will manipulate the data based on the needs of the users.

5.9.2. Start-up and Shut down

These are boundary conditions which are concerning the start up and shut down circumstances of system components. Even though we handled many of those conditions in earlier steps, we decided to construct these following three boundary use cases in order to reduce their high complexity.

Start-Stop Interface	Whenever the Student starts the program, the BBsimInterface component is started with a Mainframe, as Student interacts with the created interface system, the BBsimInterface activates Transition subsystem. The interface is not shut down until the program is exited.

[1] SSH stands for Secure Shell.

Start-Stop Design Even though the circuit design step is possible to be skipped, if Student continues to do it after creation of Project, CircuitDesign component is started with one of three options which is chosen by Student. Those three options are: DrawTruthTable - filling output values in the truth table; DrawKarnaughMap – completing KMap values; GenerateLogicFunctions – selecting 1's from KMaps and generating functions for outputs. The CircuitDesign is stopped when those subsections are either confirmed as done or skipped by Student.

Start-Stop Project Management Project Management starts at the very beginning of the program by default, while giving student two choices: create project and load project. Other entry points for this boundary condition are when Student wants to load another project and when he wants to save the current project. For each case, Project Management is shut down after the process –whichever is, create, load or save- is completed. In case of permission errors, this use case invokes Check Permission.

5.9.3. Exception Handling

Exceptions are best examples to fit for definitions of boundary. They are rare and extreme cases that still needed to be considered. It is quite difficult to write code that is free of bugs or exceptions. However, though, we mostly deal with exception handling in minor situations in which the possible exception can be avoided with simple code segments. The ones listed below are more complex ones which require more sophisticated solutions, as boundary use cases.

Check Permission This use case is invoked by project management. In the case that the directory specified by Student is not permitted to be written on, an error is risen and control is given to this use case. In the flow of events, first the program warns user that specified path is not available, and then it suggests the path of current user of the system.

Print Failed There can be several causes for a printing process to interrupt. Connection between printer and computer can disappear or printing can have some hardware related problem and so on. In those cases, program invokes this use case. At this point, program looks to the print transaction logs and ensures the security of unaccomplished tasks to be queued until the problem is resolved.

Connection Down This is the use case that takes control when the connection is broken while fetching help data from online server. As for event flow, the use case wraps the so far retrieved help and provides user both options of using wrapped online data and of using offline data for help.

6. OBJECT DESIGN

6.1. Design Patterns

In the context of the project, **Abstract Factory**, **Builder** and **Strategy** design patterns are used. The decision behind selecting these patterns and the place that we used them are explained in the subsections below.

6.1.1: Strategy Design Pattern

Problem Description : In the Drawing Circuit Schematic part of the program, there are different circuit components which are input, output, twoInputGate and oneInputGate. To generate outputs of whole circuit schematic to corresponding inputs, it is needed to calculate outputs of circuit components separately. Although the aim is same and to calculate output of a component, classes of these components have different behaviours, algorithms, to compute outputs. The problem is that we do not want to use several conditional statements, and we want to have a single class to switch on different algorithms. The aim is to increase modifiability of the program.

Solution : Deal with different behaviours of the same problem in different classes. It means to split different algorithms into different objects.

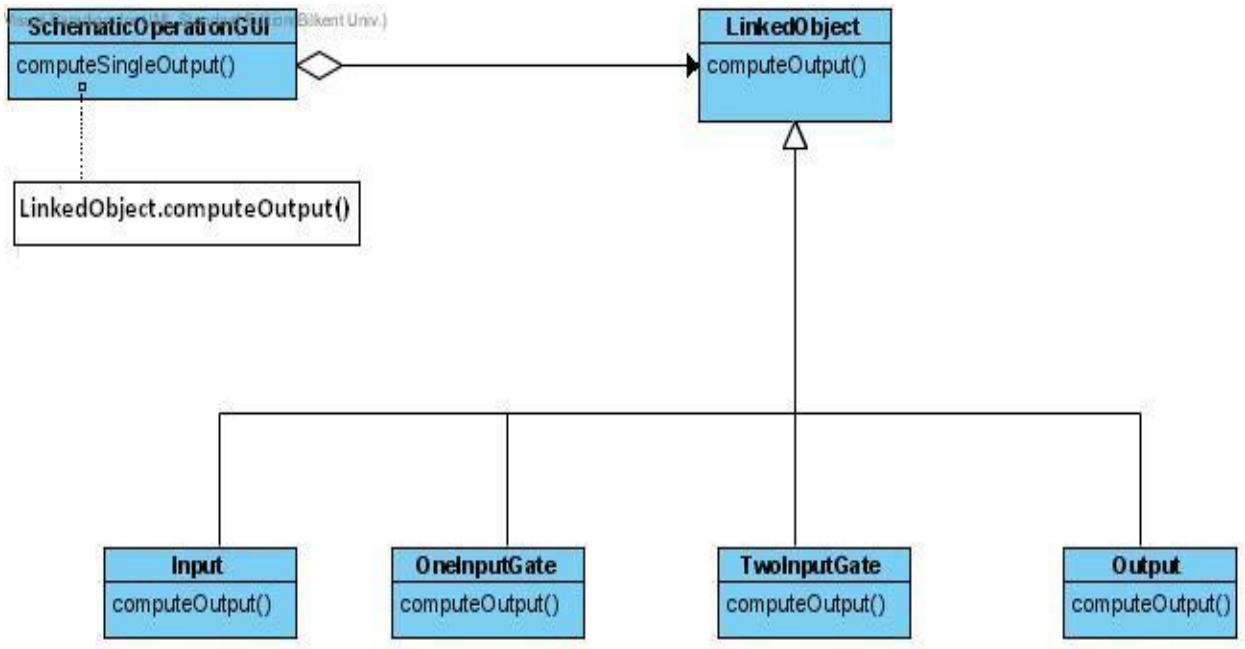


Figure 59 : Strategy Design Pattern

Consequences :

a) Strategy pattern makes it easy to add new components which their outputs are needed to be computed in a different way. In other words, it increases modifiability of the program.

b) SchematicOperationGUI handles output computation of 4 different circuit components by a single line of code in method computeSingleOutput.

c) Abstract class LinkedObject is used as an interface between circuit components and SchematicOperationGUI class.

d) Strategy pattern eliminates use of several conditional statements.

e) Stragety pattern provides to benefit from various implementations of the same problem.

f) As a disadvantage, strategy pattern causes increase in number of objects.

6.1.2. Adapter Design Pattern

Problem Description : We have an ANDGate class. We want to use that class by using abstract class LinkedObject. The aim is to have a common reusable interface for different gate objects. However, interface of ANDGate class does not match the one we need. Our problem is to use ANDGate class by a common interface for gate objects regardless of ANDGate interface is compatible or not.

Solution : The solution is to create a class that cooperates with ANDGate class which does not necessarily have compatible interface.

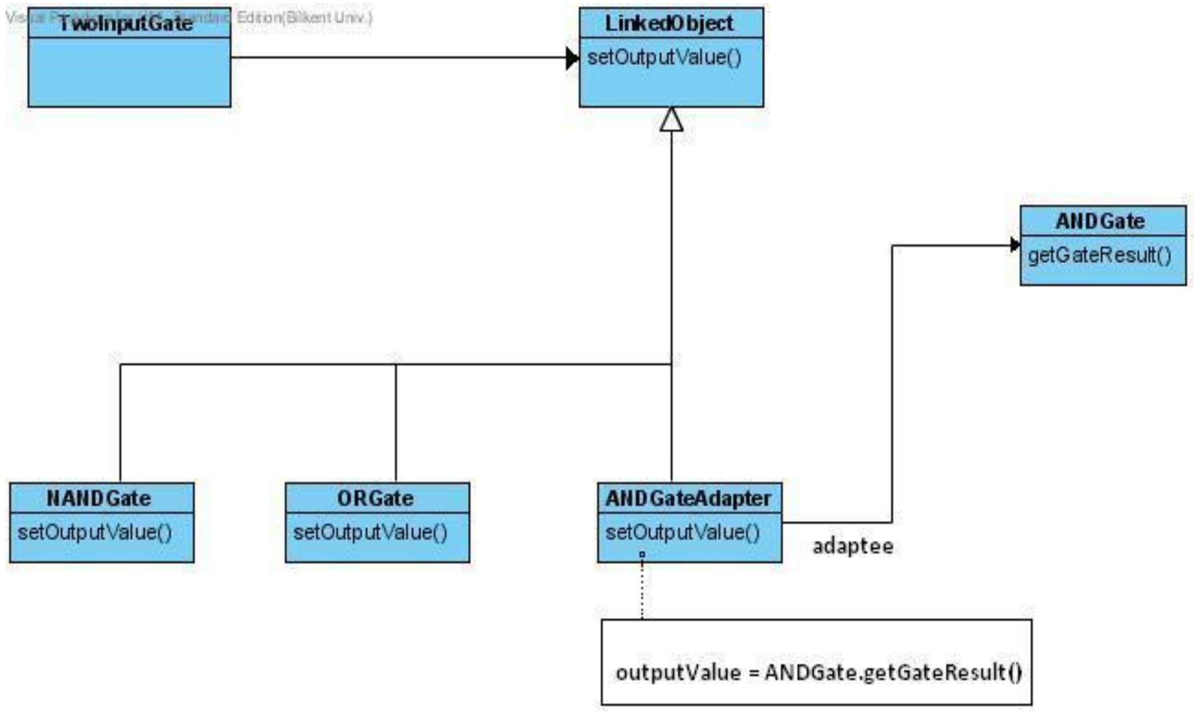


Figure 60 : Adapter Design Pattern

Consequences :

a) It became easy to add new types of gates which their interface are not compatible with common interface.

b) Adapter design pattern makes it hard to override Adaptee class' behaviour.

6.1.3. Facade Design Pattern

Problem Description : Convert all data storage classes into a different package and let one class to deal with those classes so that user interface classes fetch and update the data using provided methods.

Solution : XMLStorage class does the read and write operations for the project class and project class has attributes as different classes. ProjectControl class provides an API with its methods that can deal with the operations of XMLStorage class.

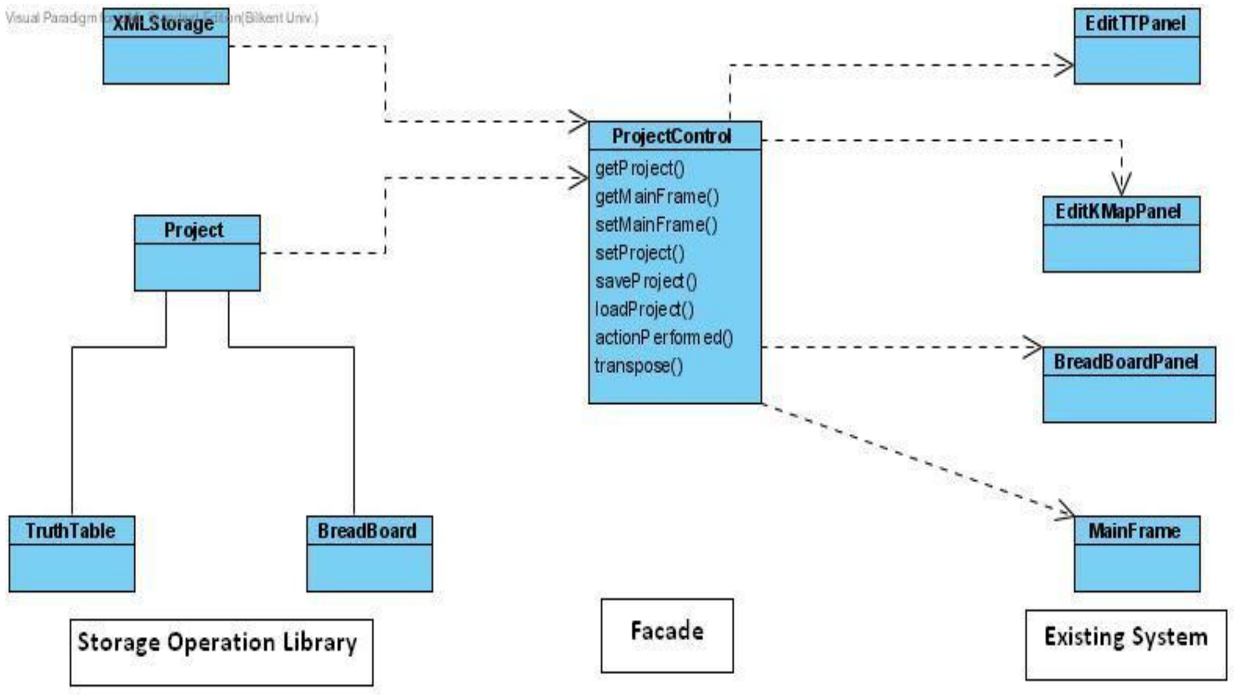


Figure 61 : Façade Design Pattern

Consequences :

a) User interface classes, like MainFrame, do not need to know XMLStorage class' operations.

b) User interface classes do not need to know the changes in the design of classes which are the attributes of Project class.

c) A well defined API is provided by ProjectControl class, which is Facade class in this case, so that a complicated API sets including different classes.

d) Developer can change the design of XMLStorage class or attribute classes, without changing embedded codes inside user interface classes. Since ProjectControl operations are used by the user interface, acknowledgement of ProjectControl about the changes is enough.

e) Dependencies of outside code is reduced, since one single class is used to cooperate.

6. 2. Class Interfaces

Inheritance and Delegation conventions are explained and the interface of every single class is provided below.

LinkedObject is an **abstract** class that represents circuit schematic components such as inputs, gates and outputs. It is the super class of **Gate** and **IO** classes.

LinkedObject
linkedObjectIcon : ImageIcon # iconX : int # iconY : int # iconWidth : int # iconHeight : int # linkedObjectType : int # outputValue : int # firstInputIsUsed : boolean # secondInputIsUsed : boolean
+ setIcon(icon : ImageIcon) + getIcon() : ImageIcon + setIconX (X : int) + getIconX() : int + setIconY (Y: int) + getIconY() : int + setIconWidth(width : int) + getIconWidth() : int + setIconHeight(height : int) + getIconHeight() : int + getType() : int + setInputOneUse(check : boolean) + setInputTwoUse(check2 : boolean) + inputOneIsUsed() : boolean + inputTwoIsUsed() : boolean + setOutputValue() + getOutputValue() : int *+ computeOutput*(wireList : ArrayList<Wire>)

The following methods are important for LinkedObject class;

+setInputOneUse(check : boolean)	sets first input of any circuit component as used or not.
+setInputTwoUse(check2 : boolean)	sets second input of component as used or not. It is used only for two input gates because there is no other component that has two inputs.

Gate is an **abstract** class that represents all gates used in circuit schematic. It is the super class of **TwoInputGate** and **OneInputGate** classes.

Gate
#outputX : int # outputY : int
+ setOutputLocation(X : int, Y : int) + getOutputX() : int + getOutputY() : int

TwoInputGate is a class that represents gates that takes two inputs. It is the super class of **ANDGate**, **ORGate**, **NANDGate**, **NORGate**, **XORGate** and **XNORGate** classes.

TwoInputGate
#upperInputX : int #upperInputY : int #lowerInputX : int #lowerInputY : int #inputValue1 : int # inputValue2 : int

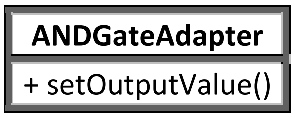
+ setInputLocations(upperX : int, upperY : int, lowerX : int, lowerY : int)
+ setInputValue1(in1 : int)
+ setInputValue2(in2 : int)
+ getUpperInputX() : int
+ getUpperInputY() : int
+ getLowerInputX() : int
+ getLowerInputY() : int
+ computeOutput(wireList : ArrayList<Wire>)

OneInputGate is a class that represents gates that takes one input. It is the super class of **NOTGate** class.

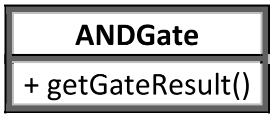
OneInputGate
inputX : int # inputY : int # inputValue : int

+ setInputLocation(X : int, Y : int)
+ setInputValue(in : int)
+ getInputX() : int
+ getInputY() : int
+ computeOutput(wireList : ArrayList<Wire>)

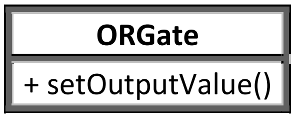
ANDGateAdapter is a class that represents two input AND gate of the circuit schematic.



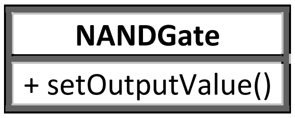
ANDGate is a class that represents two input AND gate of the circuit schematic.



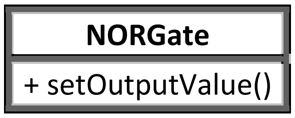
ORGate is a class that represents two input OR gate of the circuit schematic.



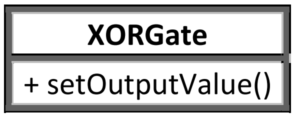
NANDGate is a class that represents two input NAND gate of the circuit schematic.



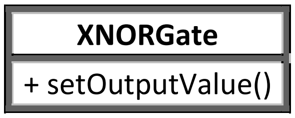
NORGate is a class that represents two input NOR gate of the circuit schematic.



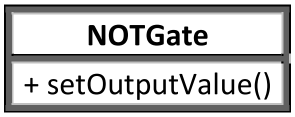
XORGate is a class that represents two input XOR gate of the circuit schematic.



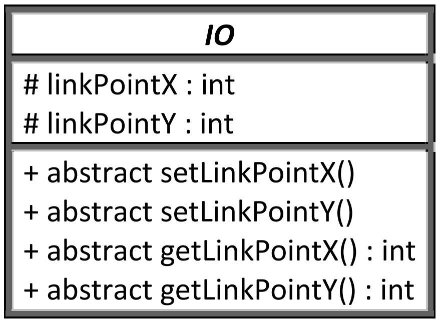
XNORGate is a class that represents two input XNOR gate of the circuit schematic.



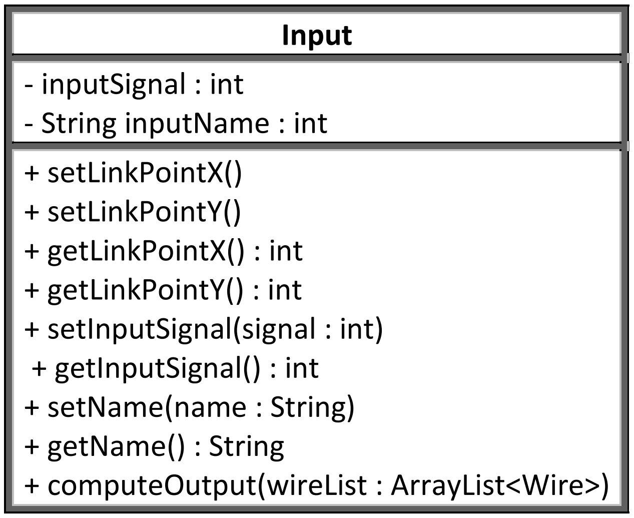
NOTGate is a class that represents NOT gate of the circuit schematic.



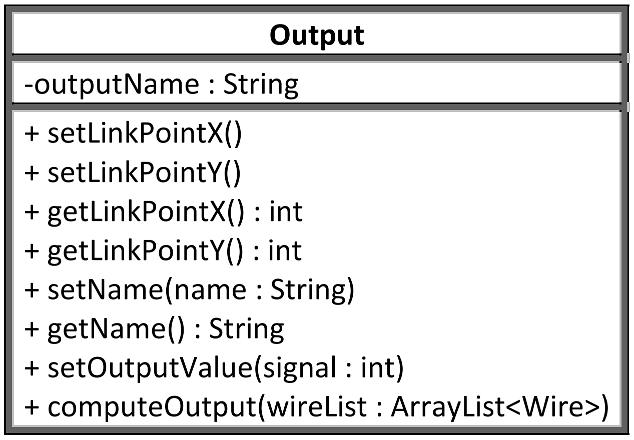
IO is an **abstract** class that represents input and output components of the circuit schematic. Since many properties of inputs and outputs come from **LinkedObject** class, only properties X and Y position values of the point that an input or output is linked to any gate is come from **IO** class. It is the super class of **Input** and **Output** classes.


IO
# linkPointX : int
# linkPointY : int
+ abstract setLinkPointX()
+ abstract setLinkPointY()
+ abstract getLinkPointX() : int
+ abstract getLinkPointY() : int

Input is a class that represen ts an input component of the circuit schematic.



Output is a class that represents an output component of the circuit schematic.



SchematicIO is a class that represents collection of inputs and outputs in the circuit schematic.

The following methods are important for **SchematicIO** class;

+ setInputs(panelHeight : int, iconArr : ImageIcon[])	sets icons, icon positions, icon sizes and link points of all inputs in the schematic.
+ setOutputs(panelWidth : int , panelHeight : int, iconArr : ImageIcon[])	sets icons, icon positions, icon sizes and link points of all outputs.

SchematicIO
- inputArr : Input[] - outputArr : Output[] - inputNumber : int - outputNumber : int - inputNames : String[]

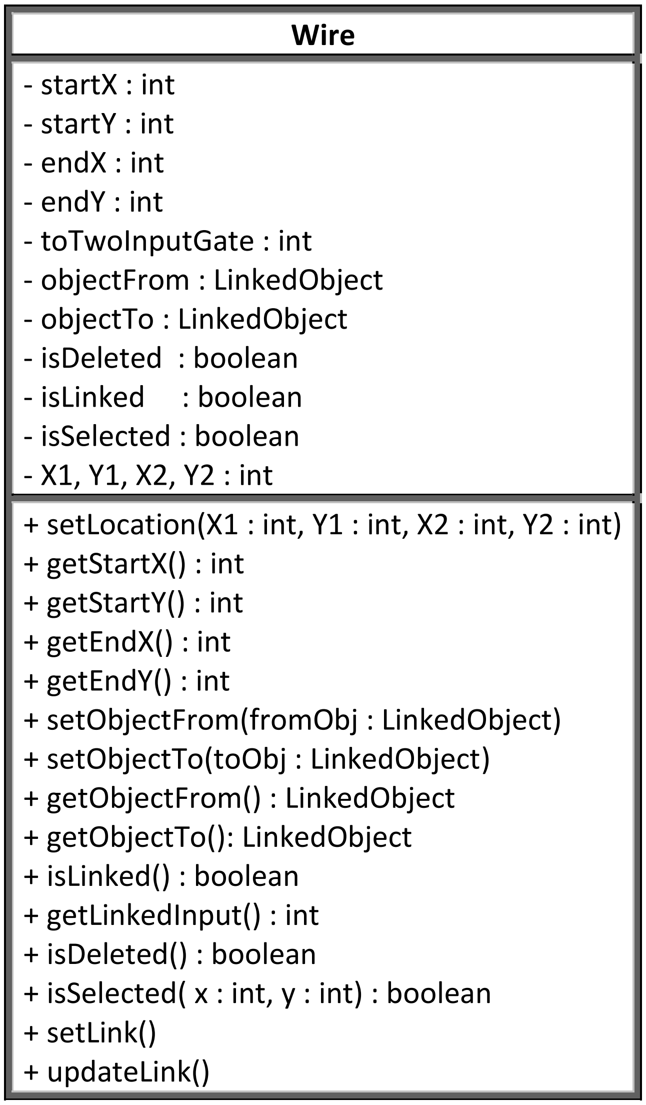
+ getInputNumber() : int
+ getOutputNumber() : int
+ setInputs(panelHeight : int, iconArr : ImageIcon[])
+ setOutputs(panelWidth : int , panelHeight : int, iconArr : ImageIcon[])
+ setInputNames()
+ setInputValues(inputs : int[])
+ getInputArr() : Input[]
+ getOutputArr() : Output[]

Wire is aclass that represents wire component which connects inputs, gates and outputs to each other on the circuit schematic.

The following methods are important for **Wire** class;

+ setObjectFrom(fromObj : LinkedObject)	sets component which start point of wire is linked.
+ setObjectTo(toObj : LinkedObject)	sets component which end point of wire is linked.
+ getObjectFrom() : LinkedObject	returns component which start point of wire is linked.

+ getObjectTo(): LinkedObject	returns component which end point of wire is linked.
+ getLinkedInput() : int	returns the gate input which wire is linked. It is used for two input gates. If upper input is linked, then it returns 1, otherwise it returns 2.



Wire

- startX : int
- startY : int
- endX : int
- endY : int
- toTwoInputGate : int
- objectFrom : LinkedObject
- objectTo : LinkedObject
- isDeleted : boolean
- isLinked   : boolean
- isSelected : boolean
- X1, Y1, X2, Y2 : int

+ setLocation(X1 : int, Y1 : int, X2 : int, Y2 : int)
+ getStartX() : int
+ getStartY() : int
+ getEndX() : int
+ getEndY() : int
+ setObjectFrom(fromObj : LinkedObject)
+ setObjectTo(toObj : LinkedObject)
+ getObjectFrom() : LinkedObject
+ getObjectTo(): LinkedObject
+ isLinked() : boolean
+ getLinkedInput() : int
+ isDeleted() : boolean
+ isSelected(x : int, y : int) : boolean
+ setLink()
+ updateLink()

SchematicGUI is a class that represents user interface of the circuit schematic. All operations and user interface components such as buttons, panels are collected in **SchematicGUI**.

SchematicGUI
- schGui : SchematicOperationGUI - buttonPanel : JPanel - ANDButton : JButton - ORButton : JButton - NANDButton : JButton - NORButton : JButton - XORButton : JButton - XNORButton : JButton - NOTButton : JButton - wireButton : JButton - RemoveButton : JButton - ComputeButton : Jbutton - inputLabels : Jlabel[] - outputLabels : Jlabel[] - inputBoxes:JCombobox[] - schematicInput : int - outputNames : String []
+ actionPerformed(e : ActionEvent)

SchematicOperationGUI is class that represents the user interface about adding, removing and linking components. Additionally, output calculations are made in that class.

The following methods are important for **SchematicOperationGUI** class;

+ computeWholeSystem(inOut : SchematicIO)	computes output values of the circuit schematic.
+computeSingleOutput(element: LinkedObject)	computes output of a single component on the circuit schematic.

SchematicOperationGUI
- gateList : ArrayList <Gate>
- wireList : ArrayList <Wire>
- inOut : SchematicIO
- inputIconArr : ImageIcon[]
- outputIconArr : ImageIcon[]
- panelWidth : int
- panelHeight : int
- componentIcon : ImageIcon
- componentImage : Image
- linkCount, pressedGate : int
- failureInSystem : boolean
- removeStatus : boolean
+ paintComponent(g : Graphics)
+ addGateToList(myGate : Gate)
+ addWireToList(myWire : Wire)
+ getWireList() : ArrayList<Wire>
+ setRemoveStatus(bool : boolean)
+ getSchematicIO() : SchematicIO
+ computeWholeSystem(inOut : SchematicIO)
+ computeSingleOutput(element : Output)
+ mouseClicked(e : MouseEvent)
+ mousePressed(e : MouseEvent)
+ mouseReleased(e : MouseEvent)
+ mouseDragged(e : MouseEvent)

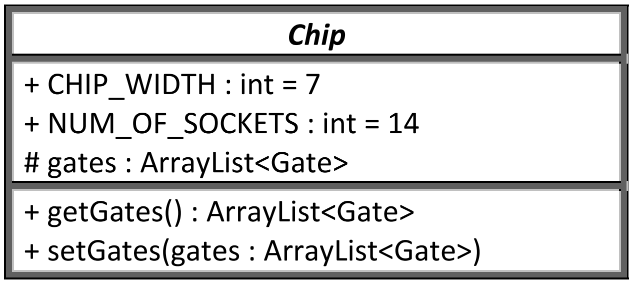
Switchable class is **abstract** and provides a basis for all the switchable objects in the project. The **state** of the object – i.e. whether switched ON or OFF – is kept.

Switchable
- state: boolean
+ getState() : boolean
+ setState(state : boolean)

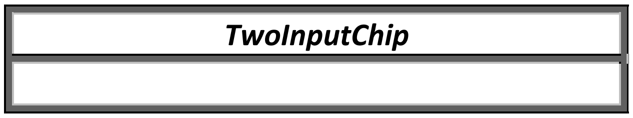
Locatable is an interface that enforces having a **position** for the objects that implements itself.

Locatable
+ getPosition() : Point
+ setPosition(point : Point)

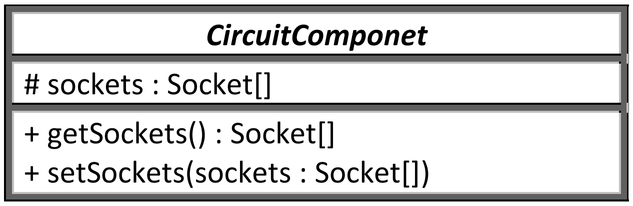
Chip is an **abstract** class that extends **CircuitComponent** class. It provides a basis for all of the **Chip** objects in the project and keeps the **gates** in these **Chip** objects.


Chip
+ CHIP_WIDTH : int = 7
+ NUM_OF_SOCKETS : int = 14
# gates : ArrayList<Gate>
+ getGates() : ArrayList<Gate>
+ setGates(gates : ArrayList<Gate>)

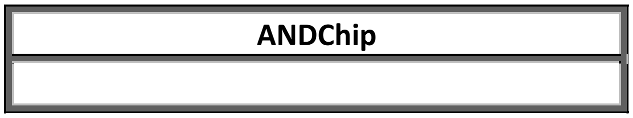
TwoInputChip is an **abstract** class that extends **Chip** class. It is just a placeholder for all of the Chip objects that has 2 different inputs.



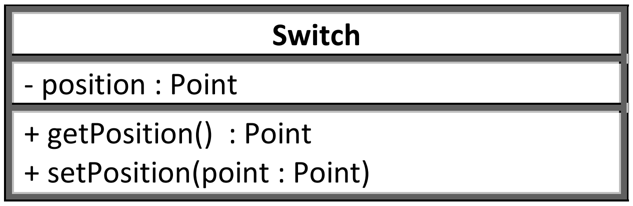
CircuitComponent is an **abstract** class that provides a basis for the objects having **Socket** pins.



ANDChip class **extends TwoInputChip** class and is a placeholder for all the ANDChip classes.



Switch class **extends Switchable** class and **implements Locatable interface**. It is being used to set the input values of the **Breadboard** class from the **Workstation** class.



Led class **extends Switchable** class and **implements Locatable interface**. It is being used to see the output results of the input values from the **Breadboard** class to the **Workstation** class.

Led
- position : Point
+ getPosition() : Point + setPosition(point : Point)

Wire class **extends CircuitComponent** class**.** It is being used to connect the **CircuitComponent**s to the **Input** and **Output Socket**s. Every **Wire** object has a unique **Color** that is provided by the *generateColor()* method.

Wire
- color : Color
+ getColor() : Color + setColor(color : Color) + generateColor() : Color

Socket class **extends Switchable** class and **implements Locatable interface**. It has a *connectedCoponent* that holds the **CircuitComponent** object that is connected to every particular instance of the **Socket** class.

Socket
+ GAP : int = 20 + SIDE : int = 10; - position : Point - connectedComponent : CircuitComponent

+ getPosition() : Point
+ setPosition(point : Point)
+ getConnectedComponent() : CircuitComponent
+ setConnectedComponent(connectedComponent : CircuitComponent)

Breadboard class **implements Locatable interface**. It lists all the **Socket** objects, **Wire** objects and **Chip** objects that are connected to each other on the **Breadboard**.

Breadboard
+ BREADBOARD_ROWNUMBER : int = 14 + BREADBOARD_COLNUMBER : int = 30 + LAST_CHIP_POSITION : int = 23 - position : Point - sockets : ArrayList<Socket> - chips : ArrayList<Chip> - wires : ArrayList<Wire>
- initializeSocketsWithPosition(numOfSocket : int) + addChip(sockets : Socket[]) : boolean + removeChip(positionX : int) : boolean + addWire(wire : Wire, whichEnd : String, firstEnd : boolean) : boolean + removeWire(wire : Wire, whichEnd : String) : boolean + checkSocket(socketPosition : Point) : boolean + connectSocketTo(socketPosition : Point, component : CircuitComponent) : boolean + disconnectSocket(socketPosition : Point) : boolean + getSocket(socketPosition : Point) : Socket + getSocketsForChip(positionX : int) : Socket[] + setSocketsForChip(positionX : int, chip : Chip) + resetSocketsForChip(positionX : int) + getPosition() : Point + setPosition(point : Point) + getSockets() : ArrayList<Socket> + setSockets(sockets : ArrayList<Socket>) + getChips() : ArrayList<Chip> + setChips(chips : ArrayList<Chip>) + getWires() : ArrayList<Wire> + setWires(wires : ArrayList<Wire>)

Workstation class **implements Locatable interface**. It lists all the **Wire** objects, and **Input**, **Output**, groud and power **Socket** objects that are connected to each other on **Workstation** or **Breadboard** object.

Workstation
- position : Point - wires : ArrayList<Wire> - inputs : ArrayList<Socket> - outputs : ArrayList<Socket> - ground : Socket - power : Socket
+ initializeIO() + initializeIO(numOfInputs : int, numOfOutputs : int) + addWire(wire : Wire, whichEnd :String, firstEnd : boolean) : boolean + removeWire(wire : Wire, whichEnd : String) : boolean + connectSocketTo(socketPosition : Point, component : CircuitComponent) : boolean + disconnectSocket(socketPosition : Point) : boolean + getSocket(socketPosition : Point) : Socket + getPosition() : Point + setPosition(point : Point) + setWires(wires : ArrayList<Wire>) + getWires() : ArrayList<Wire> + getInputs() : ArrayList<Socket> + setInputs(inputs : ArrayList<Socket>) + getOutputs() : ArrayList<Socket> + setOutputs(outputs : ArrayList<Socket>) + getGround() :Socket + setGround(ground : Socket) + getPower() : Socket + setPower(power : Socket)

BuildBBControl class controls the whole process of building the circuit on **Breadboard** and **Workstation**. To do this, it needs to have a **BBEditPanel** object.

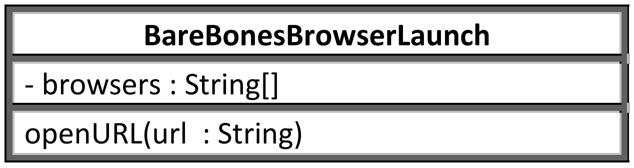
BuildBBControl
- breadboard: Breadboard - workstation : Workstation - editPanel : BBEditPanel
+ addWire(beginX : int, beginY : int, endX : int, endY : int) : boolean + removeWire(positionX : int, positionY : int) : boolean

BBEditPanel class **extends JPanel** class of Java’s Swing library. It is where the user interacts with the program to draw the circuit design on **Breadboard** and **Workstation**. This panel delegates the **Breadboard** and **Workstation** objects from the **BuildBBControl** class.

BBEditPanel
- breadboard: Breadboard - workstation : Workstation - breadboardLabel : JLabel - workstationLabel : JLabel - socketLabels : ArrayList<JLabel> - inputLabels : ArrayList<JLabel> - switchLabels : ArrayList<JLabel> - outputLabels : ArrayList<JLabel> - ledLabels : ArrayList<JLabel> - groundLabel : JLabel - powerLabel : JLabel - images : ImageIcon[]

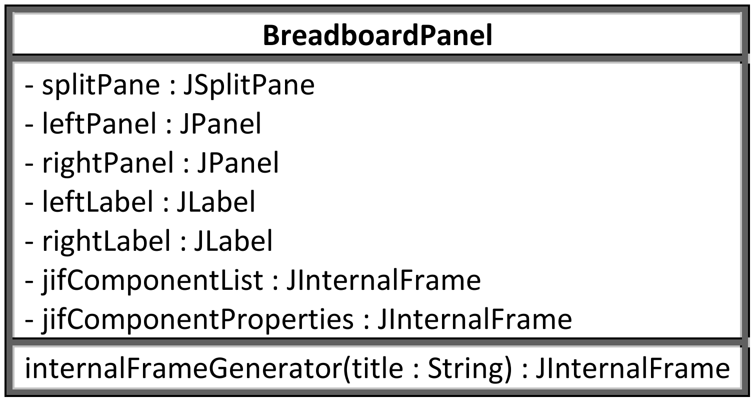
+ initializeLabel(icon : ImageIcon, positionX : int, positionY : int) : JLabel
+ initializeSocketLabels()
+ initializeInputAndSwitchLabels()
+ addComponents()
+ paint(g : Graphics)
+ setBreadboard(breadboard : Breadboard)
+ getBreadboard() : Breadboard
+ setWorkstation(workstation : Workstation)
+ getWorkstation() : Workstation
+ setImages(images : ImageIcon[])
+ getImages() : ImageIcon[]

BareBonesBrowserLaunch class provides a static method that launches the default browser of operating systems accordingly, using the URL given as a parameter to the method. In this project, this class is used to get online help.



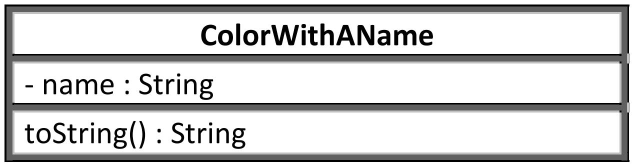
```
BareBonesBrowserLaunch
- browsers : String[]
openURL(url  : String)
```

BreadboardPanel class extends **JPanel** class of Java's Swing library. It provides a framework for the layout of the drawing panels.



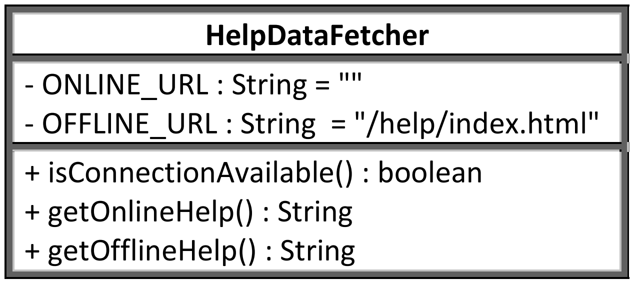
```
BreadboardPanel
- splitPane : JSplitPane
- leftPanel : JPanel
- rightPanel : JPanel
- leftLabel : JLabel
- rightLabel : JLabel
- jifComponentList : JInternalFrame
- jifComponentProperties : JInternalFrame
internalFrameGenerator(title : String) : JInternalFrame
```

ColorWithAName class extends String class adding a color attribute to a normal String object. It provides a method to get string attribute of the class.



```
ColorWithAName
- name : String
toString() : String
```

HelpDataFetcher class fetches the help data from the web server of **BBSim**.



```
HelpDataFetcher
- ONLINE_URL : String = ""
- OFFLINE_URL : String  = "/help/index.html"
+ isConnectionAvailable() : boolean
+ getOnlineHelp() : String
+ getOfflineHelp() : String
```

HelpDisplayPanel extends **JPanel**. It displays the help data that was fetched by using **HelpDataFetcher** class.

HelpDisplayPanel
- html : JEditorPane - dataFetcher : HelpDataFetcher
+ createHyperLinkListener() : HyperlinkListener

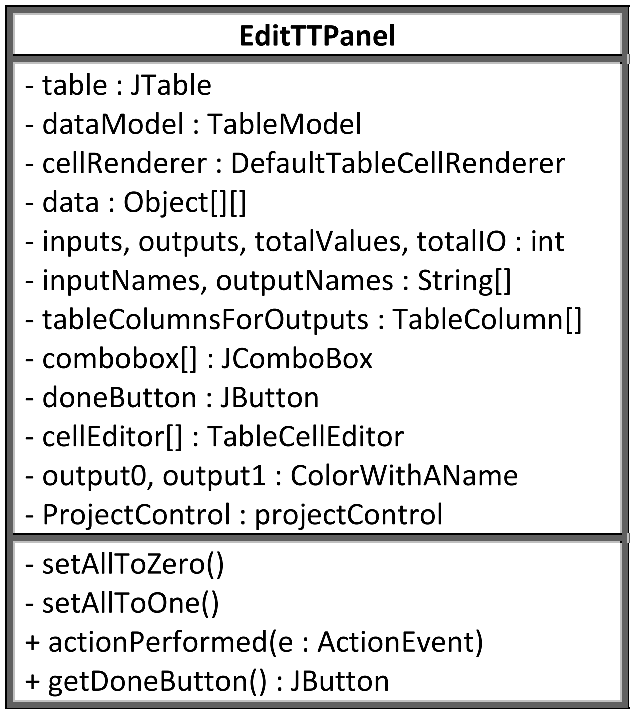
EditKMapPanel class is a panel which generates user interface for multiple number of Kmaps. It generates functions using selected 1's from the KMaps by the user. It verifies the generated functions with the values entered to the TruthTable by the user. It also has many private operations dealing with the algorithm working behind the function generation and verification processes. Among its attributes, there are the functions generated from K-Maps, selected 1's from the Kmaps, and user interface elements such as panels, labels buttons etc.

EditKMapPanel
- noOfInputs : int - noOfOutputs : int - rows : int - columns : int - inputNames : String[] - outputNames : String[] - kmapPanels : JPanel[] - kmapOuterPanels : JPanel[] - buttonPanels : JPanel[] - functionPanels : JPanel[] - doneButton : JButton - verifyButton : JButton - getFunctionButtons : JButton[] - functions : JTextField[] - outputs : JTextField[] - labels : JLabel[][][] - panels : JPanel[][][] - kms : JPanel - verifyPanel : JPanel - pane : JScrollPane - panePanel : JPanel - selected : boolean[][][]

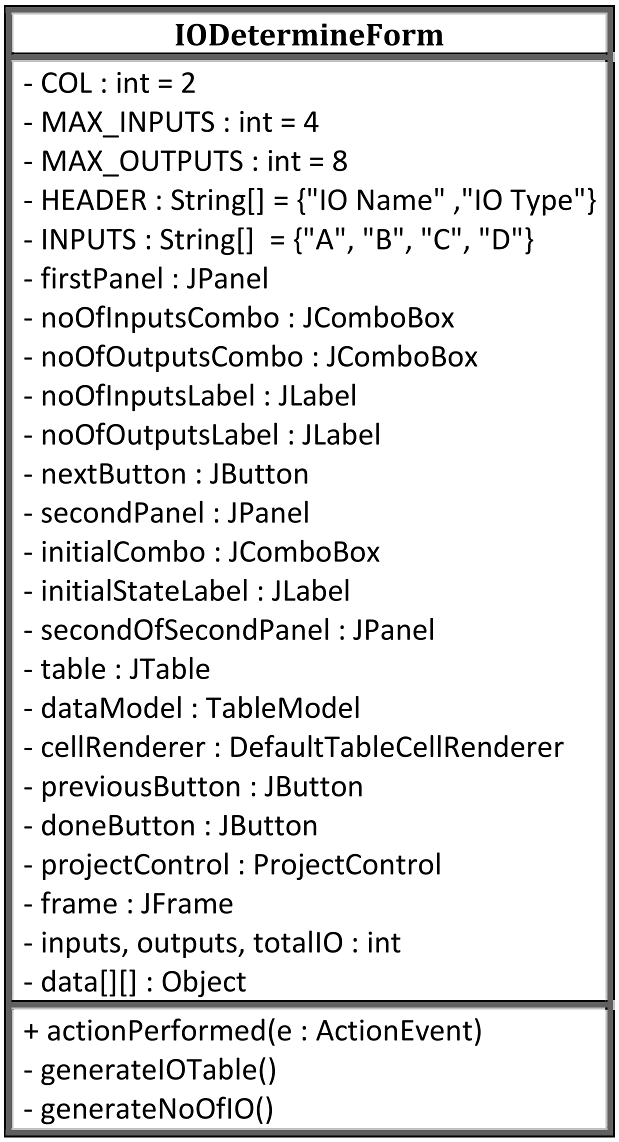
- data : int[][]
- LABEL_SIZE : int = 50
- COLORS : Color[]
- colorPointer : int[]
- selectedBorder : Border
- unselectedBorder : Border

+ isSelectable(locations : ArrayList<Integer>) : boolean
- size2Selectable(locations : ArrayList<Integer>) : boolean
- size4Selectable(locations : ArrayList<Integer>) : boolean
- size8Selectable(locations : ArrayList<Integer>) : boolean
- numbering(i : int, j : int) : int
- getFunction(selectedItems : ArrayList<Integer>) : String
- stringExtender(input : String) : String
- functionGenerator(inputs : ArrayList<String>) : String
- getLoc(value : int) : int[]
- swapper(a : int) : int
- changeSelection(i : int, j : int, k : int)
- colorAdder(x : Color, y : Color) : Color
- colorSubtructor(x : Color, y : Color) : Color
- removeAllSelections(i : int)
- verifyWithTruthTable(functionStrings: String[], allData :int[][], inputNames :String[]) :boolean
- and(input : String) : String
- or(input : ArrayList<String>) : int
+ mouseEntered(e : MouseEvent)
+ mouseExited(e : MouseEvent)
+ mousePressed(e : MouseEvent)
+ mouseReleased(e : MouseEvent)
+ mouseClicked(e : MouseEvent)
+ actionPerformed(e : ActionEvent)

EditTTPanel class provides a user interface to make the user see the default inputs and make him set the output values accordingly. It has two operations to set output values to all 1 or all 0. It also keeps track of data visualized as table and ready to be written to the project as an attribute. The values determined in this panel forms the basis for the verification in the following steps.


EditTTPanel
- table : JTable
- dataModel : TableModel
- cellRenderer : DefaultTableCellRenderer
- data : Object[][]
- inputs, outputs, totalValues, totalIO : int
- inputNames, outputNames : String[]
- tableColumnsForOutputs : TableColumn[]
- combobox[] : JComboBox
- doneButton : JButton
- cellEditor[] : TableCellEditor
- output0, output1 : ColorWithAName
- ProjectControl : projectControl
- setAllToZero()
- setAllToOne()
+ actionPerformed(e : ActionEvent)
+ getDoneButton() : JButton

IODetermineForm class is the user interface used as one of the project creation steps. The user can determine the number of inputs and outputs together with the output names. It also informs the **ProjectControl** to create project according to the information determined using this panel.



IODetermineForm

- COL : int = 2
- MAX_INPUTS : int = 4
- MAX_OUTPUTS : int = 8
- HEADER : String[] = {"IO Name" ,"IO Type"}
- INPUTS : String[] = {"A", "B", "C", "D"}
- firstPanel : JPanel
- noOfInputsCombo : JComboBox
- noOfOutputsCombo : JComboBox
- noOfInputsLabel : JLabel
- noOfOutputsLabel : JLabel
- nextButton : JButton
- secondPanel : JPanel
- initialCombo : JComboBox
- initialStateLabel : JLabel
- secondOfSecondPanel : JPanel
- table : JTable
- dataModel : TableModel
- cellRenderer : DefaultTableCellRenderer
- previousButton : JButton
- doneButton : JButton
- projectControl : ProjectControl
- frame : JFrame
- inputs, outputs, totalIO : int
- data[][] : Object

+ actionPerformed(e : ActionEvent)
- generateIOTable()
- generateNoOfIO()

TruthTable class is used to store Truth Table data.

TruthTable
- table : boolean[][] - rowNumber : int - colNumber : int
+ setTable(table : boolean[][]) + getTable() : boolean[][] + getRowNumber() : int + setRowNumber(rowNumber : int) + getColNumber() : int + setColNumber(colNumber : int) + toggleCell(rowNum : int, colNum : int) : boolean + setCell(rowNum : int, colNum : int, setUp : boolean) : boolean

KMap class is used to store Karnaugh Map data.

KMap
- values : boolean[][]
+ getValueAt(row : int, col : int) : int + setValueAt(row : int, col : int, value : int)

CreditsPanel is a panel that holds the information related to developers and also their pictures. The pictures together with the information are shown as a panel.

CreditsPanel
- bbsimString : String[] - mustafaString : String[] - korpeString : String[] - battalString : String[] - salimString : String[] - bbsimColors : Color[] - mustafaColors : Color[] - korpeColors : Color[] - battalColors : Color[] - salimColors : Color[] - bbsimPicPath : String - mustafaPicPath : String - korpePicPath : String - battalPicPath : String - salimPicPath : String - titlePanel : JPanel - title : JLabel - picturePanel : JPanel - mustafa : JLabel - korpe : JLabel - battal : JLabel - salim : JLabel - infoPanel : JPanel - picture : ImageIcon - infoPicPanel : JPanel - pictureLabel : JLabel - info2Panel : JPanel - JLabel[ ] infoLine; - backPanel : JPanel - backButton : JButton - pHolder : JPanel - font : Font

+ preparePanels()
+ setInfoPanel (details : String[], colors : Color[], pictFilePath : String)
+ mouseEntered(e : MouseEvent)
+ mouseExited(e : MouseEvent)
+ mousePressed(e : MouseEvent)
+ mouseReleased(e : MouseEvent)
+ mouseClicked(e : MouseEvent)

ProjectControl class is the main control object that controls the overall project using both data and user interface classes. It is also the **facade** class of the applied facade pattern. As attributes, it has **MainFrame** object to manage the user interface classes and Project object to manage data classes. It also has an access to **XMLStorage** class for save and load operations.

ProjectControl
- project : Project - mainframe : MainFrame

+ getProject() : Project
+ getMainFrame() : MainFrame
+ setMainFrame(mainFrame : MainFrame)
+ setProject(project : Project)
+ saveProject()
+ loadProject(url : String)
+ actionPerformed(e : ActionEvent)
- transpose(int[][] data) : int[][]

Project class is the data class that keeps the data required to save a project to enable user to load the project using the file saved before. It keeps the data in private attributes and it also provides getter and setter operations to manage all these attributes.

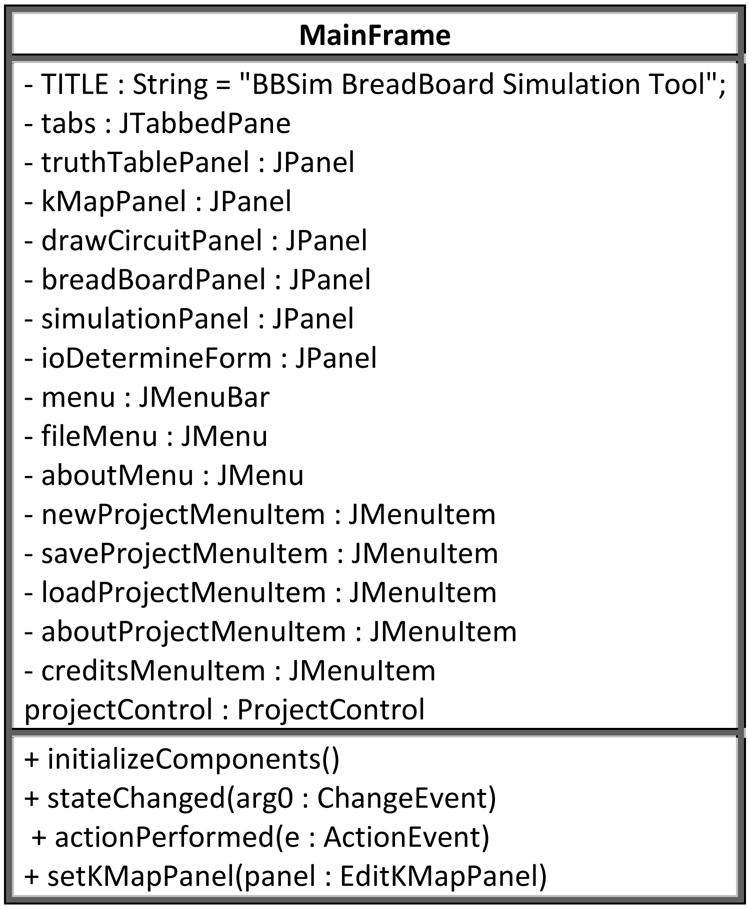
Project
- noOfInputs, noOfOutputs : int - truthTable : TruthTable - functions : String[] - outputNames : String[] -inputNames : String[] - cs : CircuitSchematic - bb : BreadBoard - gates : ArrayList<Gate> - initialState : int - currentState : int - url : String
+ getInitialState() : int + getCurrentState() : int + getNoOfInputs() : int + getNoOfOutputs() : int + getTruthTable() : TruthTable

+ getFunctions() : String[]
+ getOutputNames() : String[]
+ getInputNames() : String[]
+ getCs() : CircuitSchematic
+ getBb() : BreadBoard
+ getGates() : ArrayList<Gate>
+ getUrl() : String
+ setUrl(url : String)
+ setNoOfInputs(noOfInputs : int)
+ setNoOfOutputs(noOfOutputs : int)
+ setTruthTable(truthTable : TruthTable)
+ setFunctions(functions : String[])
+ setOutputNames(outputNames : String[])
+ setInputNames(inputNames : String[])
+ setCs(cs : CircuitSchematic)
+ setBb(bb : BreadBoard)
+ setGates(gates : ArrayList<Gate>)
+ setInitialState(initialState : int)
+ setCurrentState(currentState : int)

XMLStorage class has two static methods, one of which is loading an existing project from a file and the other is saving the current open project to a file.

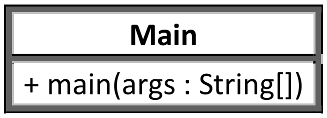
XMLStorage
+ saveProject(project : Project, url : String) : boolean + loadProject(url : String) : Project

MainFrame class keeps all the panels of user interface. The panels are put ina tabbed fashion. It has a menu that provides operations for managing the project like creating, saving loading. It also enables users to get help online. It listens the actions of menu items and make **ProjectControl** know the actions and handle them.



```
MainFrame
- TITLE : String = "BBSim BreadBoard Simulation Tool";
- tabs : JTabbedPane
- truthTablePanel : JPanel
- kMapPanel : JPanel
- drawCircuitPanel : JPanel
- breadBoardPanel : JPanel
- simulationPanel : JPanel
- ioDetermineForm : JPanel
- menu : JMenuBar
- fileMenu : JMenu
- aboutMenu : JMenu
- newProjectMenuItem : JMenuItem
- saveProjectMenuItem : JMenuItem
- loadProjectMenuItem : JMenuItem
- aboutProjectMenuItem : JMenuItem
- creditsMenuItem : JMenuItem
projectControl : ProjectControl
+ initializeComponents()
+ stateChanged(arg0 : ChangeEvent)
 + actionPerformed(e : ActionEvent)
+ setKMapPanel(panel : EditKMapPanel)
```

Main class is the runnable class of overall project. It creates **ProjectContol** and **MainFrame** objects to run a new program. It also delegates **ProjectControl** to **MainFrame** to provide a bidirectional association.



```
Main
+ main(args : String[])
```

6.3. Specifying Contracts using OCL

```
// 1. To create an EditKMapPanel there should be a Project and all truth table values
// should be either 0 or 1
context EditKMapPanel inv:
        project <> nil and project.truthTable->forAll(row | row->forAll(col | col = 0 or col = 1));

// 2. The number given as parameter should be 2 or 3,
// since gray code is iterating like 0 1 3 2
context EditKMapPanel::swapper(number1) pre:
        number1 = 2 or number1 = 3

// 3. Since the K-Map can be 4*4 at most,
// parameter value converted to a grid location should be between 0 and 16
context EditKMapPanel::getLoc(value) pre:
        value < 16 and value >= 0

// 4. Size of parameter inputs should be greater than 0 to generate a function from that
context EditKMapPanel::functionGenerator(inputs) pre:
        inputs->size() > 0

// 5. Each color pointer (specific to k-map) should point to values between 0
// and the size of color array.
context EditKMapPanel::actionPerformed(event) pre:
        self.colorPointer->forAll(v | v >= 0 or v < self.COLORS->size())

// 6. Increment the color pointer of corresponding output by one if possible,
// if not(i.e. it reaches to the end of constant color array) then equate it to 0.
context EditKMapPanel::actionPerformed(event) post:
        self.colorPointer->select(a | self.outputs->at(a).isSelected())
                                        -> forAll(v | if(@pre.v = self.COLORS->size() - 1) v >= 0
                                        else v = @pre.v+1 endif)

// 7. Set borders of all labels to unselected
context EditKMapPanel::removeAllSelections(i:Integer) post:
        self.labels->forAll(i | i->forAll(j | j->forAll(k | k.border = self.unselectedBorder)))
```

// **8.** Length of input parameter should be smaller than number of inputs
// since the method adds 0 to the beginning until it reaches to the number of inputs
context EidtKMapPanel::stringExtender(input:String) pre:
    input.length() <= self.noOfInputs

// **9.** Length of returned result should be equal to the number of inputs
context EidtKMapPanel::stringExtender(input:String) post:
    result.length() = self.noOfInputs

// **10.** In order to talk about a selection, there should be at least one item selected.
context EditKMapPanel::isSelectable(locations:ArrayList) pre:
    locations->size() > 0

// **11.** All outputs should be set to 0
context EditTTPanel::setAlltoZero() post:
    self.table.getModel()->forAll(row | row->forAll(column = self.output0))

// **12.** All outputs should be set to 1
context EditTTPanel::setAlltoOne() post:
    self.table.getModel()->forAll(row | row->forAll(column = self.output1))

// **13.** Input should consist of 1s and 0s
context Verify::and(input) pre:
    input->forAll(index | index = 0 or index = 1)

// **14.** Output is either 1 or zero
context Verify::and(input) post:
    result = 1 or result = 0

// **15.** Input should consist of 1s and 0s
context Verify::or(input) pre:
    input->forAll(index | index = 0 or index = 1)

// **16.** Output is either 1 or zero
context Verify::or(input) post:
    result = 1 or result = 0

// **17.** Result color should be the total of color1 and color2
context EditKMapPanel::colorAdder(color1, color2) post:
    result.red = color1.red + color2.red and
    result.blue = color1.blue + color2.blue and
    result.green = color1.green + color2.green and
    result.alpha = color1.alpha + color2.alpha

// **18.** To save the project, project should not be nil and url should be valid.
context XMLStorage::saveProject(project:Project, url:String) pre:
    project <> nil and isValidUrl(url)

// **19.** To load the project url should be valid
context XMLStorage::loadProject(url:String) pre:
    isValidUrl(url)

// **20.** Sets locations of inputs
context TwoInputGate::setInputLocations(upperX,upperY,lowerX,lowerY) post:
    self.upperInputX = upperX and
    self.upperInputY = upperY and
    self.lowerInputX = lowerX and
    self.lowerInputY = lowerY

// **21.** input value can be set to either 0 or 1
context TwoInputGate::setInputValue1(in1) pre:
    in1 = 0 or in1 = 1

// **22.** input value is set to the parameter after execution.
context TwoInputGate::setInputValue1(in1) post:
    self.inputValue1 = in1

// **23.** wireList should not be empty to compute corresponding output.
context TwoInputGate::computeOutput(wireList) pre:
    not wireList->isEmpty()

// **24.** To create a MainFrame object there should be a ProjectControl object.
context MainFrame inv:
    projectControl <> null

// **25.** To save a project there should exist a project.
```
context ProjectControl::saveProject() pre:
        self.getProject() <> nil
```

// **26.** To load a project, given url should be valid.
```
context ProjectControl::loadProject(url:String) pre:
        XMLStorage.isValidUrl(url)
```

// **27.** MainFrame of ProjectControl should not be null, since it provides the user interaction
```
context ProjectControl::setMainFrame(mainFrame:MainFrame) pre:
        mainFrame <> nil
```

// **28.** There should be at least one input
```
context Project::setInputNames(inputNames) pre:
        inputNames->size() > 0
```

// **29.** Input value is set to the parameter after execution
```
context Project::setInputNames(inputNames) post:
        self.inputNames = inputNames
```

// **30.** There should be at least one output
```
context Project::setOutputNames(outputNames) pre:
        outputNames->size() > 0
```

// **31.** Output value is set to the parameter after execution
```
context Project::setOutputNames(outputNames) post:
        self.outputNames = outputNames
```

// **32.** If one of the input is 0 result is 0, 1 otherwise
```
context ANDGate::setOutputValue() post:
        if(self.inputValues->includes(0))
                self.outputValue = 0
        else
                self.outputValue = 1
        endif
```

```
// **33.** If one of the input is 1 result is 1, 0 otherwise
context ORGate::setOutputValue() post:
        if(self.inputValues->includes(1))
                self.outputValue = 1
        else
                self.outputValue = 0
        endif

// **34.** Output should be located into the screen (i.e. location should be inside the panel)
context Gate::setOutputLocation(x, y) pre:
        x >= 0 and y >= 0

// **35.** Output locations are set to the parameters after execution
context Gate::setOutputLocation(x, y) post:
        self.outputX = x and self.outputY = y
```

7. CONCLUSION

Our project is called Logic Design and Breadboard Simulator in long, but we also call it BBSim which stands for Breadboard Simulator. As it is understood by the name, what system mainly does is to help user to make logic design of circuits and simulate them on a virtual breadboard.

In that project, we intended to handle problems which we faced in the course CS223. Most of the Computer Science Departments of Colleges offer course(s) about Digital Circuit design and implementation. Assignments and Projects given in such courses involve designing, creating and testing digital systems where interfacing of input and output devices is occasionally necessary. However, using such kind of devices has some limitations as;

- The interfacing Workstations are only available in the department laboratories. So, it is not possible to work on the given assignment or project outside of the laboratories.
- The given tasks require a direct interaction with electronic hardware, such as Logic Gates, Wires, and Batteries. Frequently, these hardware components break or run out. Under such circumstances, Students are not able to test their design to detect the mistakes before handling the given task.
- Such courses are generally mandatory in Computer Science Departments; so lots of Students are involved in them. While designing, implementing and testing the given task, Students working in laboratories have to be supervised by experienced Students or Staff. Because of that, the available experienced Student or Staff to help Students is limited. Therefore, there is a time limitation on the provided work sessions.

We have faced these problems in CS 223 – Digital Design course last year. Therefore, as mentioned, we focused on developing a system which makes logic design of circuits and simulate them on a virtual breadboard to eliminate previously stated problems.

Our development of project is completed in three main steps which are project analysis, system design and object design.

The first step was the project analysis. In that step, we explained requirements analysis and analysis models. In the requirements analysis, overview of the project, functional and non-functional requirements, constraints, scenarios, use case models and user interfaces are included. Together with that, analysis object model and dynamic model of the system are mentioned in the analysis models part. Object model consists of domain lexicon and class diagrams. Apart from that, state charts and sequence diagrams are included in dynamic model of the system.

In the next step, system design of the project is handled. In that phase, we have worked on design goals, subsystem decomposition, architectural patterns and other system design activities which are hardware/software mapping, persistent data management, handling access control and security, global software control and boundary conditions. As first step, we have determined design goals which are about ease of use, reliability, robustness, portability, maintainability and documentation for open source development. After that, we had subsystem decomposition process. We handled subsystem decomposition process in two parts. In the first part, we applied basic level subsystem decomposition. However, in the second part, we described all subsystems in detail. During decomposition process, we always consider on issues low coupling and high cohesion. We divided whole system into three main parts which are interface, internal application and storage. The aim of such construction is to make whole system more resilient to changes and to make implementation stage easier. As next step, we applied two architectural patterns to our subsystem decomposition. Since we have three main parts of whole system, we had three-tier architectural pattern. Separately, since we have a central data storage mechanism, we applied a repository architectural pattern. Apart from architectural pattern issue, we worked on hardware/software mapping as next step. In that phase, we mapped our program into some hardware components. As another step, we determined access control policy and authentication process. After that, we described our approach to global software control. Due to its appropriateness to object oriented programming, we decided to select *Event-driven control* as a control flow mechanism. As last step of system design process, we handled boundary conditions.

Apart from system design issue, we worked on object design of the project. In that step, we determined whole objects of the system and their relationships. Additionally, we applied required design patterns. At the end, we completed the design of the project and eliminated the erroneous parts to the degree that it is possible.

However, the final part of the project – implementation part – was not that much successful because the time for preparing Object Design and Implementation was limited. Also, some other lectures that we have been attending separately required us to complete some other assignments. Obviously, these assignments also entail some time to complete. This was the most significant obstacle for us.

Working on this kind of project, we have learned that the design is the most important thing for Object Oriented Software development. We have learned how an ongoing process is dependent on the processes that we have completed before. We have understood how important it is to work on a synchronized project team in which all the members are aware of the progression of the project. Besides we have gained communication and teamwork skills. We encountered dilemmas and problems, and overcoming them required a fast and accurate communication between group members. Another communication practice was preparing the reports, documenting each stage of our work and all the details of the current state of the project. Technical report writing skills are also precious for an engineer. We gained experience in report writing with the help of this project.

As a final word, we are very thankful to our Instructor Bedir Tekinerdoğan and our course assitant Elif Demirli for their help and support at every step of the development of our project. Without their help, it would not be possible to complete this project and deliver it on time.

REFERENCES

1. Object-Oriented Software Engineering: Using UML, Patterns and Java, 2/E
Bernd Bruegge, Adjunct, Carnegie Mellon University
Allen H. Dutoit, Technical University of Munich
ISBN-10: 0130471100
ISBN-13: 9780130471109
Publisher: Prentice Hall
Copyright: 2004
Format: Cloth; 800 pp
Published: 09/25/2003

2. Applying UML and Patterns 2E: Introduction to Object-Oriented Analysis, Design, Unified Process, Craig Larman

APPENDIX

The full code of the project is provided below.

```
/*******************************************************************************
An abstract class that represents circuit schematic components such as inputs, gates
and outputs.
It is the super class of Gate and IO classes.
*******************************************************************************/

import java.awt.*;
import javax.swing.*;
import java.util.*;

public abstract class LinkedObject
{
	//Properties
	protected ImageIcon linkedObjectIcon;
	protected int iconX;
	protected int iconY;
	protected int iconWidth;
	protected int iconHeight;
	protected int linkedObjectType;
	protected int outputValue;

	protected boolean firstInputIsUsed;
	protected boolean secondInputIsUsed;


	//Methods
	public void setIcon(ImageIcon icon)
	{
		linkedObjectIcon = icon;
	}

	public ImageIcon getIcon()
	{
		return linkedObjectIcon;
	}

	public void setIconX(int X)
	{
		iconX = X;
	}

	public int getIconX()
	{
		return iconX;
	}

	public void setIconY(int Y)
	{
		iconY = Y;
	}

	public int getIconY()
	{
		return iconY;
	}

	public void setIconWidth(int width)
	{
		iconWidth = width;
	}

	public int getIconWidth()
	{
		return iconWidth;
	}

	public void setIconHeight(int height)
	{
		iconHeight = height;
	}

	public int getIconHeight()
	{
		return iconHeight;
	}

	public int getType()
	{
		return linkedObjectType;
	}

	public void setInputOneUse(Boolean bool)
	{
		firstInputIsUsed = bool;
	}

	public void setInputTwoUse(Boolean bool)
	{
		secondInputIsUsed = bool;
	}

	public boolean inputOneIsUsed()
	{
		return firstInputIsUsed;
	}

	public boolean inputTwoIsUsed()
	{
		return secondInputIsUsed;
	}

	public void computeOutput(ArrayList<Wire> myList)
	{};


	public void setOutputValue()
	{};


	public int getOutputValue()
	{
		return outputValue;
	}
}
```

```
/*********************************************************************************
An abstract class that represents all gates used in circuit schematic. It is the super
class of TwoInputGate and OneInputGate classes.
**********************************************************************************/

public abstract class Gate extends LinkedObject
{
       //Properties
       protected int outputX;
       protected int outputY;

       //Methods
       public void setOutputLocation(int X, int Y)
       {
              outputX = X;
              outputY = Y;
       }

       public int getOutputX()
       {
              return outputX;
       }

       public int getOutputY()
       {
              return outputY;
       }
}
/**********************************************************************************
A class that represents gates that takes two inputs. It is the super class of
ANDGateAdapter, ORGate, NANDGate, NORGate, XORGate and XNORGate classes.
**********************************************************************************/

import java.util.*;

public class TwoInputGate extends Gate
{
       //Properties
       protected int upperInputX;
       protected int upperInputY;

       protected int lowerInputX;
       protected int lowerInputY;

       protected int inputValue1;
       protected int inputValue2;


       //Constructor
       public TwoInputGate()
       {
              super();
              upperInputX = 0;
              upperInputY = 0;

              lowerInputX = 0;
              lowerInputY = 0;

              linkedObjectType = 3;

              inputValue1 = -1;
              inputValue2 = -1;
```

```
        firstInputIsUsed  = false;
        secondInputIsUsed = false;

        outputValue = -1;
}





//Methods
public void setInputLocations(int upperX, int upperY, int lowerX, int lowerY)
{
        upperInputX = upperX;
        upperInputY = upperY;

        lowerInputX = lowerX;
        lowerInputY = lowerY;
}

public void setInputValue1(int in1)
{
        inputValue1 = in1;
}

public void setInputValue2(int in2)
{
        inputValue2 = in2;
}

public int getUpperInputX()
{
        return upperInputX;
}

public int getUpperInputY()
{
        return upperInputY;
}

public int getLowerInputX()
{
        return lowerInputX;
}

public int getLowerInputY()
{
        return lowerInputY;
}

public void computeOutput(ArrayList<Wire> wireList)
{
        int index = -1;
        boolean test = false;
        boolean completed = false;

        for(int i = 0; i < wireList.size(); i++)
        {
                if(wireList.get(i).getObjectTo() == this)
                {
                        wireList.get(i).getObjectFrom().computeOutput(wireList);


                        index = i;

                        if(test == false)
                        {
        setInputValue1(wireList.get(index).getObjectFrom().getOutputValue());
                                test = true;
                        }
                        else
                        {
        setInputValue2(wireList.get(index).getObjectFrom().getOutputValue());
                                completed = true;
                        }

                }
            }

            if(completed)
            {
                completed = false;
                System.out.println("ERDINC");
                this.setOutputValue();
            }
            else
            {
                outputValue = -1;
            }
        }
}


/*****************************************************************************
A class that represents gates that takes one input. It is the super class of NOTGate
class.
*****************************************************************************/
import java.util.*;

public class OneInputGate extends Gate
{
        //Properties
        protected int inputX;
        protected int inputY;

        protected int inputValue;

        //Constructor
        public OneInputGate()
        {
            super();
            inputX = 0;
            inputY = 0;

            linkedObjectType = 4;
            inputValue = -1;

            firstInputIsUsed  = false;
            secondInputIsUsed = false;
            outputValue = -1;
        }


        //Methods
        public void setInputLocation(int X, int Y)
        {
            inputX = X;
            inputY = Y;
        }

        public void setInputValue(int in)
        {
            inputValue = in;
        }

        public int getInputX()
        {
            return inputX;
        }

        public int getInputY()
        {
            return inputY;
        }


        public void computeOutput(ArrayList<Wire> wireList)
        {
            int index = -1;

            for(int i = 0; i < wireList.size(); i++)
            {
                if(wireList.get(i).getObjectTo() == this)
                {
                        wireList.get(i).getObjectFrom().computeOutput(wireList);
                        index = i;


        setInputValue(wireList.get(index).getObjectFrom().getOutputValue());
                }
            }

            if(index != -1)
            {
                this.setOutputValue();
            }
        }
}


/*****************************************************************************
A class that Is used as an adapter to make ANDGate class to be used by common
interface.
*****************************************************************************/
class ANDGateAdapter extends TwoInputGate
{
        //Constructor
        public ANDGateAdapter()
        {
            super();
        }
```

```
	//Methods
	public void setOutputValue()
	{
		outputValue = ANDGate.getGateResult(inputValue1, inputValue2, 
outputValue);
	}
}



/*********************************************************************************
A class that represents two input AND gate of the circuit schematic. It is bind to 
common interface by 
ANDGateAdapter class.
*********************************************************************************/
class ANDGate
{
	//Methods
	public static int getGateResult(int input1, int input2, int output)
	{
		if(input1 == 0 && input2 == 0 )
		{
			output = 0;
		}
		else if(input1 == 1 && input2 == 0 )
		{
			output = 0;
		}
		else if(input1 == 0 && input2 == 1 )
		{
			output = 0;
		}
		else if(input1 == 1 && input2 == 1 )
		{
			output = 1;
		}
		else
		{
			output = -1;
		}

		return output;

	}
}









/*********************************************************************************
A class that represents two input OR gate of the circuit schematic.
*********************************************************************************/
import java.awt.*;
```

```
class ORGate extends TwoInputGate
{
	//Constructors
	public ORGate()
	{
		super();
	}

	//Methods
	public void setOutputValue()
	{
		if(inputValue1 == 0 && inputValue2 == 0)
		{
			outputValue = 0;
		}
		else if(inputValue1 == 0 && inputValue2 == 1)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 1 && inputValue2 == 0)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 1 && inputValue2 == 1)
		{
			outputValue = 1;
		}
		else
		{
			outputValue = -1;
		}
	}
}


/******************************************************************************
A class that represents two input NAND gate of the circuit schematic.
******************************************************************************/
import java.awt.*;

class NANDGate extends TwoInputGate
{
	//Constructors
	public NANDGate()
	{
		super();
	}

	//Methods
	public void setOutputValue()
	{
		if(inputValue1 == 0 && inputValue2 == 0)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 0 && inputValue2 == 1)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 1 && inputValue2 == 0)
		{
			outputValue = 1;
		}
		else if( inputValue1 == 1 && inputValue2 == 1)
		{
			outputValue = 0;
		}
		else
		{
			outputValue = -1;
		}
	}
}



/*********************************************************************************
A class that represents two input NOR gate of the circuit schematic.
*********************************************************************************/
import java.awt.*;

class NORGate extends TwoInputGate
{
	//Constructors
	public NORGate()
	{
		super();
	}


	//Methods
	public void setOutputValue()
	{
		if(inputValue1 == 0 && inputValue2 == 0)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 0 && inputValue2 == 1)
		{
			outputValue = 0;
		}
		else if(inputValue1 == 1 && inputValue2 == 0)
		{
			outputValue = 0;
		}
		else if(inputValue1 == 1 && inputValue2 == 1)
		{
			outputValue = 0;
		}
		else
		{
			outputValue = -1;
		}
	}

}
```

```
/*********************************************************************************
A class that represents two input XOR gate of the circuit schematic.
*********************************************************************************/
import java.awt.*;

class XORGate extends TwoInputGate
{
	//Constructors
	public XORGate()
	{
		super();
	}

	//Methods
	public void setOutputValue()
	{
		if(inputValue1 == 0 && inputValue2 == 0)
		{
			outputValue = 0;
		}
		else if(inputValue1 == 0 && inputValue2 == 1)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 1 && inputValue2 == 0)
		{
			outputValue = 1;
		}
		else if(inputValue1 == 1 && inputValue2 == 1)
		{
			outputValue = 0;
		}
		else
		{
			outputValue = -1;
		}
	}

}


/*********************************************************************************
A class that represents  two input XNOR gate of the circuit schematic.
*********************************************************************************/
import java.awt.*;

class XNORGate extends TwoInputGate
{
	//Constructors
	public XNORGate()
	{
		super();
	}

	//Methods
	public void setOutputValue()
	{
		if(inputValue1 == 0 && inputValue2 == 0)
		{
			outputValue = 1;
			}
			else if(inputValue1 == 0 && inputValue2 == 1)
			{
				outputValue = 0;
			}
			else if(inputValue1 == 1 && inputValue2 == 0)
			{
				outputValue = 0;
			}
			else if(inputValue1 == 1 && inputValue2 == 1)
			{
				outputValue = 1;
			}
			else
			{
				outputValue = -1;
			}
		}

}



/*********************************************************************************
A class that represents NOT gate of the circuit schematic.
*********************************************************************************/
public class NOTGate extends OneInputGate
{
		//Constructor
		public NOTGate()
		{
			super();
		}

		//Methods
		public void setOutputValue()
		{
			if(inputValue == 0)
			{
				outputValue = 1;
			}
			else if(inputValue == 1)
			{
				outputValue = 0;
			}
			else
			{
				outputValue = -1;
			}
		}
}



/*********************************************************************************
An abstract class that represents input and output components of the circuit schematic.
Since many properties of inputs and outputs come from LinkedObject class, only
properties X and Y position values of the point that an input or output is linked to
any gate is come from IO class. It is the super class of Input and Output classes.
*********************************************************************************/
public abstract class IO extends LinkedObject
{
	//Properties
	protected int linkPointX;
	protected int linkPointY;


	//Methods
	public abstract void setLinkPointX();

	public abstract void setLinkPointY();

	public abstract int getLinkPointX();

	public abstract int getLinkPointY();
}




/*******************************************************************************
A class that represents an input component of the circuit schematic.
*******************************************************************************/
import java.util.*;

public class Input extends IO
{
	//Properties
	private int inputSignal;
	private String inputName;

	//Constructor
	public Input()
	{
		super();
		linkedObjectType = 1;
		inputSignal = 0;

		firstInputIsUsed  = false;
		secondInputIsUsed = false;

		outputValue = -1;
	}

	//Methods
	public void setLinkPointX()
	{
		linkPointX = iconX + iconWidth;
	}

	public void setLinkPointY()
	{
		linkPointY = iconY + (iconHeight/2);
	}

	public int getLinkPointX()
	{
		return linkPointX;
	}

	public int getLinkPointY()
	{
		return linkPointY;
	}
	public void setInputSignal(int signal)
	{
		inputSignal = signal;
	}

	public int getInputSignal()
	{
		return inputSignal;
	}

	public void setName(String name)
	{
		inputName = name;
	}

	public String getName()
	{
		return inputName;
	}

	public void computeOutput(ArrayList<Wire> wireList)
	{
		outputValue = inputSignal;
	}
}
```

```
/*************************************************************************************
A class that represents an output component of the circuit schematic.
************************************************************************************/im
port java.util.*;

public class Output extends IO
{
	//Properties
	private String outputName;

	//Constructor
	public Output()
	{
		super();
		linkedObjectType = 2;

		firstInputIsUsed  = false;
		secondInputIsUsed = false;

		outputValue = -1;
	}

	//Methods
	public void setLinkPointX()
	{
		linkPointX = iconX;
	}

	public void setLinkPointY()
	{
		linkPointY = iconY + (iconHeight/2);
	}
	public int getLinkPointX()
	{
		return linkPointX;
	}

	public int getLinkPointY()
	{
		return linkPointY;
	}

	public void setName(String name)
	{
		outputName = name;
	}

	public String getName()
	{
		return outputName;
	}

	public void setOutputValue(int signal)
	{
		if(signal == 0 || signal == 1)
			outputValue = signal;
		else
			outputValue = -1;
	}

	public void computeOutput(ArrayList<Wire> wireList)
	{
		int index = -1;

		if(wireList != null)
		{
			for(int i = 0; i < wireList.size(); i++)
			{
				if(wireList.get(i).getObjectTo() == this)
				{
	wireList.get(i).getObjectFrom().computeOutput(wireList);
					index = i;

	setOutputValue(wireList.get(index).getObjectFrom().getOutputValue());
				}
			}
		}
	}
}


/*********************************************************************************
A class that represents collection of inputs and outputs in the circuit schematic.
*********************************************************************************/

import java.awt.*;
import javax.swing.*;

public class SchematicIO
{
	//Properties
```

```
private Input[]  inputArr;
private Output[] outputArr;

private int inputNumber;
private int outputNumber;

private String[] inputNames;


//Constructor
public SchematicIO(int numberOfInputs, int numberOfOutputs, String[] names)
{
       inputArr    = new Input[numberOfInputs];
       outputArr   = new Output[numberOfOutputs];

       inputNumber  = numberOfInputs;
       outputNumber = numberOfOutputs;

       for(int i = 0; i < inputNumber; i++)
       {
              inputArr[i] = new Input();
       }

       for(int k = 0; k < outputNumber; k++)
       {
              outputArr[k] = new Output();
       }

       for(int t = 0; t < outputNumber; t++)
       {
              outputArr[t].setName(names[t]);
       }

       inputNames = new String[4];

       inputNames[0] = "A";
       inputNames[1] = "B";
       inputNames[2] = "C";
       inputNames[3] = "D";
}


//Methods
public int getInputNumber()
{
       return inputNumber;
}

public int getOutputNumber()
{
       return outputNumber;
}

public void setInputs(int panelHeight, ImageIcon[] iconArr)
{
       if(iconArr != null)
       {
              int partial = panelHeight / (inputNumber + 1);


              for(int i = 0; i < inputNumber; i++)
              {
```

```
                        inputArr[i].setIcon(iconArr[i]);
                        inputArr[i].setIconX(10);
                        inputArr[i].setIconY((i+1)* partial);
                        inputArr[i].setIconWidth(iconArr[i].getIconWidth());
                        inputArr[i].setIconHeight(iconArr[i].getIconHeight());
                        inputArr[i].setLinkPointX();
                        inputArr[i].setLinkPointY();
                    }
                }
        }

        public void setOutputs(int panelWidth, int panelHeight, ImageIcon[] iconArr)
        {
                if(    iconArr != null)
                {
                        int partial = panelHeight / (outputNumber + 1);

                        for(int i = 0; i < outputNumber; i ++)
                        {
                                outputArr[i].setIcon(iconArr[i]);
                                outputArr[i].setIconWidth(iconArr[i].getIconWidth());
                                outputArr[i].setIconHeight(iconArr[i].getIconHeight());
                                outputArr[i].setIconX(panelWidth - 10 -
iconArr[i].getIconWidth());
                                outputArr[i].setIconY((i+1) * partial);
                                outputArr[i].setLinkPointX();
                                outputArr[i].setLinkPointY();
                        }
                }
        }

        public void setInputNames()
        {
                for(int i = 0; i < inputNumber; i++)
                {
                        inputArr[i].setName(inputNames[i]);
                }
        }

        public void setInputValues(int[] inputs)
        {
                for(int i = 0; i < inputNumber; i++)
                {
                        inputArr[i].setInputSignal(inputs[i]);
                }
        }

        public Input[] getInputArr()
        {
                return inputArr;
        }

        public Output[] getOutputArr()
        {
                return outputArr;
        }
}



/******************************************************************************
```

```
A class that represents wire component which connects inputs, gates and outputs to each
other on the circuit schematic.
*********************************************************************************/

import java.util.*;

public class Wire
{
	//Properties
	private int startX;
	private int startY;

	private int endX;
	private int endY;

	private int toTwoInputGate;				// 1--->upperInput, 2----
>lowerInput

	private LinkedObject objectFrom;
	private LinkedObject objectTo;

	private boolean isDeleted;
	private boolean isLinked;
	private boolean isSelected;

	private int X1, Y1, X2, Y2;			//Just for in-use.


	//Constructor
	public Wire()
	{
		startX = 0;
		startY = 0;

		endX = 0;
		endY = 0;

		toTwoInputGate = 0;

		isDeleted  = false;
		isLinked   = false;
		isSelected = false;
	}

	//Methods
	public void setLocation(int X1, int Y1, int X2, int Y2)
	{
		startX = X1;
		startY = Y1;
		endX   = X2;
		endY   = Y2;
	}

	public int getStartX()
	{
		return startX;
	}

	public int getStartY()
	{
		return startY;
	}
```

```
public int getEndX()
{
	return endX;
}

public int getEndY()
{
	return endY;
}

public void setObjectFrom(LinkedObject fromObj)
{
	objectFrom = fromObj;
}

public void setObjectTo(LinkedObject toObj){
	objectTo = toObj;
}

public LinkedObject getObjectFrom()
{
	return objectFrom;
}

public LinkedObject getObjectTo()
{
	return objectTo;
}

public boolean isLinked()
{
	return isLinked;
}

public int getLinkedInput()
{
	return toTwoInputGate;
}

public boolean isDeleted()
{
	return isDeleted;
}

public boolean isSelected(int x, int y)
{
	if( x >= startX && x <= endX)
	{
		if((y <= startY && y >= endY) || (y >= startY && y <= endY))
		{
			double mainDiffX = Math.abs(endX - startX);
			double mainDiffY = Math.abs(endY - startY);

			if(mainDiffY == 0)
			{
				if(y == endY && startX <= x && x <= endX)
				{
					isSelected = true;
				}
				else
					isSelected = false;
```

```
                        }
                        else
                        {
                              double mainRatio = mainDiffY / mainDiffX;

                              double diffX = Math.abs(endX - x);
                              double diffY = Math.abs(endY - y);

                              double ratio = diffY / diffX;

                              if(mainRatio - ratio < 0.2)
                              {
                                    isSelected = true;
                              }
                              else
                                    isSelected = false;
                        }
                  }
                  else
                  {
                        isSelected = false;
                  }
            }
            else
            {
                  isSelected = false;
            }

            return isSelected;
      }

public void setLink()
{
      if(objectFrom != null && objectTo != null)
      {

            //Input - OneInputGate
            if(objectFrom.getType() == 1 && objectTo.getType() == 4 &&
!objectTo.inputOneIsUsed())
            {
                  X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                  Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                  X2 = objectTo.getIconX();
                  Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                  setLocation(X1, Y1, X2, Y2);
                  isLinked = true;
                  objectTo.setInputOneUse(true);
            }

            //Input - TwoInputGate
            else if(objectFrom.getType() == 1 && objectTo.getType() == 3)
            {
                  if(objectTo.inputOneIsUsed() == false && objectTo.inputTwoIsUsed()
== false)
                  {
                        if(objectFrom.getIconY() > objectTo.getIconY())
                        {
                              X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                              Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
```

```
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() +
((4*objectTo.getIconHeight())/5) - 3;

                                setLocation(X1, Y1, X2, Y2);
                                isLinked = true;
                                objectTo.setInputTwoUse(true);
                                toTwoInputGate = 2;
                        }
                        else
                        {
                                X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                setLocation(X1, Y1, X2, Y2);
                                isLinked = true;
                                objectTo.setInputOneUse(true);
                                toTwoInputGate = 1;
                        }
                }
                else if(objectTo.inputOneIsUsed() == false &&
objectTo.inputTwoIsUsed() == true)
                {
                        X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                        Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                        X2 = objectTo.getIconX();
                        Y2 = objectTo.getIconY() + ((objectTo.getIconHeight())/5) +
2;

                        setLocation(X1, Y1, X2, Y2);
                        isLinked = true;
                        objectTo.setInputOneUse(true);
                        toTwoInputGate = 1;
                }
                else if(objectTo.inputOneIsUsed() == true &&
objectTo.inputTwoIsUsed() == false)
                {
                        X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                        Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                        X2 = objectTo.getIconX();
                        Y2 = objectTo.getIconY() + ((4*objectTo.getIconHeight())/5)
- 3;

                        setLocation(X1, Y1, X2, Y2);
                        isLinked = true;
                        objectTo.setInputTwoUse(true);
                        toTwoInputGate = 2;
                }
        }

        //TwoInputGate - Output
        else if(objectFrom.getType() == 3 && objectTo.getType() == 2 &&
!objectTo.inputOneIsUsed())
        {
                X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                X2 = objectTo.getIconX();
```

```
                    Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                    setLocation(X1, Y1, X2, Y2);
                    isLinked = true;
                    objectTo.setInputOneUse(true);
            }

            //TwoInputGate - TwoInputGate
            else if(objectFrom.getType() == 3 && objectTo.getType() == 3)
            {
                    if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX())
                    {
                            if(objectTo.inputOneIsUsed() == false &&
objectTo.inputTwoIsUsed() == false)
                            {
                                    if(objectFrom.getIconY() > objectTo.getIconY())
                                    {
                                            X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                            Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                            X2 = objectTo.getIconX();
                                            Y2 = objectTo.getIconY() +
((4*objectTo.getIconHeight())/5) - 3;

                                            setLocation(X1, Y1, X2, Y2);
                                            isLinked = true;
                                            objectTo.setInputTwoUse(true);
                                            toTwoInputGate = 2;
                                    }
                                    else
                                    {
                                            X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                            Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                            X2 = objectTo.getIconX();
                                            Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                            setLocation(X1, Y1, X2, Y2);
                                            isLinked = true;
                                            objectTo.setInputOneUse(true);
                                            toTwoInputGate = 1;
                                    }
                            }
                            else if(objectTo.inputOneIsUsed() == false &&
objectTo.inputTwoIsUsed() == true)
                            {
                                    X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                    Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                    X2 = objectTo.getIconX();
                                    Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                    setLocation(X1, Y1, X2, Y2);
                                    isLinked = true;
                                    objectTo.setInputOneUse(true);
                                    toTwoInputGate = 1;
```

```
                        }
                        else if(objectTo.inputOneIsUsed() == true &&
objectTo.inputTwoIsUsed() == false)
                        {

                                X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() +
((4*objectTo.getIconHeight())/5) - 3;

                                setLocation(X1, Y1, X2, Y2);
                                isLinked = true;
                                objectTo.setInputTwoUse(true);
                                toTwoInputGate = 2;
                        }
                }
        }

        //TwoInputGate - OneInputGate
        else if(objectFrom.getType() == 3 && objectTo.getType() == 4)
        {
                if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX() && !objectTo.inputOneIsUsed())
                {
                        X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                        Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                        X2 = objectTo.getIconX();
                        Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                        setLocation(X1, Y1, X2, Y2);
                        isLinked = true;
                        objectTo.setInputOneUse(true);
                }
        }

        //OneInputGate - Output
        else if(objectFrom.getType() == 4 && objectTo.getType() == 2 &&
!objectTo.inputOneIsUsed())
        {
                X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                X2 = objectTo.getIconX();
                Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                setLocation(X1, Y1, X2, Y2);
                isLinked = true;
                objectTo.setInputOneUse(true);
        }

        //OneInputGate - TwoInputGate
        else if(objectFrom.getType() == 4 && objectTo.getType() == 3)
        {
                if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX())
                {
                        if(objectTo.inputOneIsUsed() == false &&
objectTo.inputTwoIsUsed() == false)
                        {
                                if(objectFrom.getIconY() > objectTo.getIconY())
```

```
                                  {
                                         X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                         Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                         X2 = objectTo.getIconX();
                                         Y2 = objectTo.getIconY() +
((4*objectTo.getIconHeight())/5) - 3;

                                         setLocation(X1, Y1, X2, Y2);
                                         isLinked = true;
                                         objectTo.setInputTwoUse(true);
                                         toTwoInputGate = 2;
                                  }
                                  else
                                  {
                                         X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                         Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                         X2 = objectTo.getIconX();
                                         Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                         setLocation(X1, Y1, X2, Y2);
                                         isLinked = true;
                                         objectTo.setInputOneUse(true);
                                         toTwoInputGate = 1;
                                  }
                           }
                           else if(objectTo.inputOneIsUsed() == false &&
objectTo.inputTwoIsUsed() == true)
                           {
                                  X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                  Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                  X2 = objectTo.getIconX();
                                  Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                  setLocation(X1, Y1, X2, Y2);
                                  isLinked = true;
                                  objectTo.setInputOneUse(true);
                                  toTwoInputGate = 1;
                           }
                           else if(objectTo.inputOneIsUsed() == true &&
objectTo.inputTwoIsUsed() == false)
                           {

                                  X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                  Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                  X2 = objectTo.getIconX();
                                  Y2 = objectTo.getIconY() +
((4*objectTo.getIconHeight())/5) - 3;

                                  setLocation(X1, Y1, X2, Y2);
                                  isLinked = true;
                                  objectTo.setInputTwoUse(true);
                                  toTwoInputGate = 2;
```

```
                        }
                }
        }

                //OneInputGate - OneInputGate
                else if(objectFrom.getType() == 4 && objectTo.getType() == 4 &&
!objectTo.inputOneIsUsed())
                {
                        if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX())
                        {
                                X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                                Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                                setLocation(X1, Y1, X2, Y2);
                                isLinked = true;
                                objectTo.setInputOneUse(true);
                        }
                }

                if(!isLinked)
                {
                        objectFrom = null;
                        objectTo   = null;
                        X1 = 0;
                        Y1 = 0;
                        X2 = 0;
                        Y2 = 0;

                        setLocation(X1, Y1, X2, Y2);
                }
        }
        else
        {
                isDeleted = true;
        }
}




        public void updateLink()
        {
                if(objectFrom != null && objectTo != null)
                {

                        //Input - OneInputGate
                        if(objectFrom.getType() == 1 && objectTo.getType() == 4)
                        {
                                X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                                Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                                setLocation(X1, Y1, X2, Y2);
                        }

                        //Input - TwoInputGate
                        else if(objectFrom.getType() == 1 && objectTo.getType() == 3)
                        {
```

```
				if(toTwoInputGate == 1)
				{
					X1 = objectFrom.getIconX() + 
objectFrom.getIconWidth();
					Y1 = objectFrom.getIconY() + 
objectFrom.getIconHeight()/2;
					X2 = objectTo.getIconX();
					Y2 = objectTo.getIconY() + 
((objectTo.getIconHeight())/5) + 2;

					setLocation(X1, Y1, X2, Y2);
				}
				else if(toTwoInputGate == 2)
				{
					X1 = objectFrom.getIconX() + 
objectFrom.getIconWidth();
					Y1 = objectFrom.getIconY() + 
objectFrom.getIconHeight()/2;
					X2 = objectTo.getIconX();
					Y2 = objectTo.getIconY() + 
((4*objectTo.getIconHeight())/5) - 3;

					setLocation(X1, Y1, X2, Y2);
				}
			}

			//TwoInputGate - Output
			else if(objectFrom.getType() == 3 && objectTo.getType() == 2)
			{
				X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
				Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
				X2 = objectTo.getIconX();
				Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

				setLocation(X1, Y1, X2, Y2);
			}

			//TwoInputGate - TwoInputGate
			else if(objectFrom.getType() == 3 && objectTo.getType() == 3)
			{
				if(objectFrom.getIconX() + objectFrom.getIconWidth() < 
objectTo.getIconX())
				{
					if(toTwoInputGate == 2)
					{
						X1 = objectFrom.getIconX() + 
objectFrom.getIconWidth();
						Y1 = objectFrom.getIconY() + 
objectFrom.getIconHeight()/2;
						X2 = objectTo.getIconX();
						Y2 = objectTo.getIconY() + 
((4*objectTo.getIconHeight())/5) - 3;

						setLocation(X1, Y1, X2, Y2);
					}
					else if(toTwoInputGate == 1)
					{
						X1 = objectFrom.getIconX() + 
objectFrom.getIconWidth();
						Y1 = objectFrom.getIconY() + 
objectFrom.getIconHeight()/2;
						X2 = objectTo.getIconX();
```

```
                                        Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                        setLocation(X1, Y1, X2, Y2);
                                }
                        }
                        else
                        {
                                isDeleted = true;

                                if(toTwoInputGate == 1)
                                {
                                        objectTo.setInputOneUse(false);
                                }
                                else if(toTwoInputGate == 2)
                                {
                                        objectTo.setInputTwoUse(false);
                                }
                        }

                }

                //TwoInputGate - OneInputGate
                else if(objectFrom.getType() == 3 && objectTo.getType() == 4)
                {
                        if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX())
                        {
                                X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() +
objectTo.getIconHeight()/2;

                                setLocation(X1, Y1, X2, Y2);
                        }
                        else
                        {
                                isDeleted = true;
                                objectTo.setInputOneUse(false);
                        }
                }

                //OneInputGate - Output
                else if(objectFrom.getType() == 4 && objectTo.getType() == 2)
                {
                        X1 = objectFrom.getIconX() + objectFrom.getIconWidth();
                        Y1 = objectFrom.getIconY() + objectFrom.getIconHeight()/2;
                        X2 = objectTo.getIconX();
                        Y2 = objectTo.getIconY() + objectTo.getIconHeight()/2;

                        setLocation(X1, Y1, X2, Y2);
                }

                //OneInputGate - TwoInputGate
                else if(objectFrom.getType() == 4 && objectTo.getType() == 3)
                {
                        if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX())
                        {
                                if(toTwoInputGate == 2)
                                {
                                        X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                        Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                        X2 = objectTo.getIconX();
                                        Y2 = objectTo.getIconY() +
((4*objectTo.getIconHeight())/5) - 3;

                                        setLocation(X1, Y1, X2, Y2);
                                }
                                else if(toTwoInputGate == 1)
                                {
                                        X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                        Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                        X2 = objectTo.getIconX();
                                        Y2 = objectTo.getIconY() +
((objectTo.getIconHeight())/5) + 2;

                                        setLocation(X1, Y1, X2, Y2);
                                }
                        }
                        else
                        {
                                isDeleted = true;

                                if(toTwoInputGate == 1)
                                {
                                        objectTo.setInputOneUse(false);
                                }
                                else if(toTwoInputGate == 2)
                                {
                                        objectTo.setInputTwoUse(false);
                                }
                        }
                }

                //OneInputGate - OneInputGate
                else if(objectFrom.getType() == 4 && objectTo.getType() == 4)
                {
                        if(objectFrom.getIconX() + objectFrom.getIconWidth() <
objectTo.getIconX())
                        {
                                X1 = objectFrom.getIconX() +
objectFrom.getIconWidth();
                                Y1 = objectFrom.getIconY() +
objectFrom.getIconHeight()/2;
                                X2 = objectTo.getIconX();
                                Y2 = objectTo.getIconY() +
objectTo.getIconHeight()/2;

                                setLocation(X1, Y1, X2, Y2);
                        }
                        else
                        {
                                isDeleted = true;

                                objectTo.setInputOneUse(false);
                        }
```

```
                    }

            }
            else
            {
                    isDeleted = true;
            }
    }
}




/********************************************************************************
A class that represents the user interface about adding, removing and linking
components. Additionally, output calculations are made in that class.
********************************************************************************/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

class SchematicOperationGUI extends JPanel
{
    //Properties
    private ArrayList <Gate> gateList;
    private ArrayList <Wire> wireList;
            private     SchematicIO inOut;

    private ImageIcon[] inputIconArr;
    private ImageIcon[] outputIconArr;

    private int panelWidth;
    private int panelHeight;

    private MotionListener myListener;

    private int x, y, width, height, gateType;
    private int wireX1, wireY1, wireX2, wireY2;
    private int a, b;

    private ImageIcon componentIcon;
    private Image componentImage;

    private int linkCount, pressedGate;

    private boolean test, test2, failureInSystem;

    private boolean removeStatus;

    //Constructor
    public SchematicOperationGUI(int inputNo, int outputNo, String[] outputNames)
    {
            super();
            gateList = new ArrayList<Gate>();
            wireList = new ArrayList<Wire>();

            inOut = new SchematicIO(inputNo, outputNo, outputNames);

            inputIconArr  = new ImageIcon[inputNo];
            outputIconArr = new ImageIcon[outputNo];
```

```
		for(int m = 0; m < inputNo; m++)
		{
			inputIconArr[m] = new ImageIcon("C:\\Documents and 
Settings\\erdinç\\Desktop\\gates\\Buffer.png");
		}

		for(int n = 0; n < outputNo; n++)
		{
			outputIconArr[n] = new ImageIcon("C:\\Documents and 
Settings\\erdinç\\Desktop\\gates\\Buffer.png");
		}

		linkCount   = 0;
		pressedGate = 0;

		myListener = new MotionListener();

		addMouseListener(new Mouse());

		test2 = true;
		failureInSystem = false;
		removeStatus = false;
	}


	//Methods
	public void paintComponent(Graphics g)
	{
		super.paintComponent(g);

		//get panel height and weight
		panelWidth  = getWidth();
		panelHeight = getHeight();

		//Draw boundary lines
		g.drawLine(90, 0, 90, panelHeight-1);
		g.drawLine(panelWidth-90, 0, panelWidth-90, panelHeight-1);

		//Set inputs and outputs
		inOut.setInputs(panelHeight, inputIconArr);
		inOut.setOutputs(panelWidth, panelHeight, outputIconArr);

		inOut.setInputNames();     // Set Input Names

		//Draw Inputs
		for(int i = 0; i < inOut.getInputNumber(); i++)
		{
			componentIcon  = inOut.getInputArr()[i].getIcon();
			componentImage = componentIcon.getImage();

			x = inOut.getInputArr()[i].getIconX();
			y = inOut.getInputArr()[i].getIconY();
			width  = inOut.getInputArr()[i].getIconWidth();
			height = inOut.getInputArr()[i].getIconHeight();

			g.drawImage(componentImage, x, y, width, height, this);

			//Set Input Names
			x = inOut.getInputArr()[i].getIconX() + 
inOut.getInputArr()[i].getIconWidth()/2;
			y = inOut.getInputArr()[i].getIconY() - 20;
```

```
                    g.drawString(inOut.getInputArr()[i].getName(), x, y);
            }


            //Draw Outputs
            for(int k = 0; k < inOut.getOutputNumber(); k++)
            {
                    componentIcon  = inOut.getOutputArr()[k].getIcon();
                    componentImage = componentIcon.getImage();

                    x = inOut.getOutputArr()[k].getIconX();
                    y = inOut.getOutputArr()[k].getIconY();
                    width  = inOut.getOutputArr()[k].getIconWidth();
                    height = inOut.getOutputArr()[k].getIconHeight();

                    g.drawImage(componentImage, x, y, width, height, this);

                    //NAMES
                    x = inOut.getOutputArr()[k].getIconX() +
inOut.getOutputArr()[k].getIconWidth()/2;
                    y = inOut.getOutputArr()[k].getIconY() - 20;

                    g.drawString(inOut.getOutputArr()[k].getName(), x, y);
            }

            //Draw Gates
            for(int t = 0; t < gateList.size(); t++)
            {
                    componentIcon  = (gateList.get(t)).getIcon();
                    componentImage = componentIcon.getImage();

                    x = (gateList.get(t)).getIconX();
                    y = (gateList.get(t)).getIconY();
                    width  = (gateList.get(t)).getIconWidth();
                    height = (gateList.get(t)).getIconHeight();

                    g.drawImage(componentImage, x, y, width, height, this);
            }

            //Draw Wires
            for(int s = 0; s < wireList.size(); s++)
            {
                    wireList.get(s).updateLink();

                    if(!wireList.get(s).isDeleted())
                    {
                            wireX1 = (wireList.get(s)).getStartX();
                            wireY1 = (wireList.get(s)).getStartY();
                            wireX2 = (wireList.get(s)).getEndX();
                            wireY2 = (wireList.get(s)).getEndY();

                            g.drawLine(wireX1, wireY1, wireX2, wireY2);
                    }
                    else
                    {
                            wireList.remove(s);
                            s -= 1;
                    }
            }

    }
```

```
	public void addGateToList(Gate myGate)
	{
		gateList.add(myGate);
	}

	public void addWireToList(Wire myWire)
	{
		wireList.add(myWire);
	}

	public ArrayList<Wire> getWireList()
	{
		return wireList;
	}


	//YENI
	public ArrayList<Gate> getGateList()
	{
		return gateList;
	}

	public void setRemoveStatus(boolean bool)
	{
		removeStatus = bool;
	}

	public SchematicIO getSchematicIO()
	{
		return inOut;
	}


	public void computeWholeSystem(SchematicIO inOut)
	{
		for(int i = 0; i < inOut.getOutputNumber(); i++ )
		{
			computeSingleOutput(inOut.getOutputArr()[i]);
		}

		for(int k = 0; k < inOut.getOutputNumber(); k++ )
		{
			if(inOut.getOutputArr()[k].getOutputValue() == -1 &&
failureInSystem == false)
			{
				JOptionPane.showMessageDialog(this, "CHECK CONNECTIONS!");
				failureInSystem = true;
			}
		}
		failureInSystem = false;
	}

	public void computeSingleOutput(LinkedObject element) //STRATEGY PATTERN
	{
		element.computeOutput(wireList);
	}

	public class Mouse implements MouseListener
	{
		public void  mouseClicked(MouseEvent e)
		{
			System.out.println("Erdinc_1");
```

```
                    if(removeStatus == false)
                    {
                          if(wireList.size() >= 1)
                          {
                                System.out.println("Erdinc_2");

                                if(wireList.get(wireList.size()-1).getStartX() == 0 )
                                {
                                      System.out.println("Erdinc_3");

                                      for(int i = 0; i < gateList.size(); i++)
                                      {
                                            x = (gateList.get(i)).getIconX();
                                            y = (gateList.get(i)).getIconY();
                                            width  =
(gateList.get(i)).getIconWidth();
                                            height =
(gateList.get(i)).getIconHeight();



                                            if(linkCount == 0 && e.getX() < x +
width  && e.getX() > x  && e.getY() < y + height && e.getY() > y)
                                            {
                                                  wireList.get(wireList.size()-
1).setObjectFrom(gateList.get(i));
                                                  System.out.println("from_1");
                                                  linkCount++;
                                            }
                                            else if(linkCount == 1 &&  e.getX() < x
+ width  && e.getX() > x  && e.getY() < y + height && e.getY() > y)
                                            {
                                                  linkCount = 0;
                                                  wireList.get(wireList.size()-
1).setObjectTo(gateList.get(i));
                                                  System.out.println("To_1");
                                                  wireList.get(wireList.size()-
1).setLink();
                                                  repaint();

                                            }
                                      }

                                      for(int k = 0; k < inOut.getInputNumber(); k++)
                                      {
                                            x = inOut.getInputArr()[k].getIconX();
                                            y = inOut.getInputArr()[k].getIconY();
                                            width  =
inOut.getInputArr()[k].getIconWidth();
                                            height =
inOut.getInputArr()[k].getIconHeight();

                                            if(linkCount == 0 && e.getX() < x +
width  && e.getX() > x  && e.getY() < y + height && e.getY() > y)
                                            {
                                                  wireList.get(wireList.size()-
1).setObjectFrom(inOut.getInputArr()[k]);
                                                  System.out.println("from_2");
                                                  linkCount++;
                                            }
```

```
                                                else if(linkCount == 1 && e.getX() < x +
width  && e.getX() > x  && e.getY() < y + height && e.getY() > y)
                                                {
                                                        linkCount = 0;
                                                        wireList.get(wireList.size()-
1).setObjectTo(inOut.getInputArr()[k]);
                                                        System.out.println("To_2");
                                                        wireList.get(wireList.size()-
1).setLink();
                                                        repaint();
                                                }
                                        }

                                        for(int t = 0; t < inOut.getOutputNumber();
t++)
                                        {
                                                x = inOut.getOutputArr()[t].getIconX();
                                                y = inOut.getOutputArr()[t].getIconY();
                                                width  =
inOut.getOutputArr()[t].getIconWidth();
                                                height =
inOut.getOutputArr()[t].getIconHeight();

                                                if(linkCount == 1 &&  e.getX() < x +
width  && e.getX() > x  && e.getY() < y + height && e.getY() > y)
                                                {
                                                        linkCount = 0;
                                                        wireList.get(wireList.size()-
1).setObjectTo(inOut.getOutputArr()[t]);
                                                        System.out.println("To_3");
                                                        wireList.get(wireList.size()-
1).setLink();
                                                        repaint();

                                                }
                                        }
                                }
                        }
                }
                else   //remove operation
                {
                        //Remove gate
                        for(int i = 0; i < gateList.size(); i++)
                        {
                                x = (gateList.get(i)).getIconX();
                                y = (gateList.get(i)).getIconY();
                                width  = (gateList.get(i)).getIconWidth();
                                height = (gateList.get(i)).getIconHeight();
                                gateType = (gateList.get(i)).getType();

                                if(e.getX() < x + width  && e.getX() > x  && e.getY()
< y + height && e.getY() > y)
                                {
                                        for(int s = 0; s < wireList.size(); s++)
                                        {
                                                if(wireList.get(s).getObjectTo() ==
gateList.get(i))
                                                {
                                                        wireList.remove(s);
                                                        s -= 1;
                                                }
```

```
                            else if(wireList.get(s).getObjectFrom()
== gateList.get(i))
                            {

    if(wireList.get(s).getObjectTo().getType() == 3)
                                {

    if(wireList.get(s).getLinkedInput() == 1)
                                    {

    wireList.get(s).getObjectTo().setInputOneUse(false);
                                    }
                                    else
if(wireList.get(s).getLinkedInput() == 2)
                                    {

    wireList.get(s).getObjectTo().setInputTwoUse(false);
                                    }
                                }
                                else
                                {

    wireList.get(s).getObjectTo().setInputOneUse(false);
                                }

                                wireList.remove(s);
                                s -=1;
                            }
                        }

                        gateList.remove(i);
                        i -= 1;
                        removeStatus = false;
                        repaint();
                    }
                }
                //Remove Wire
                for(int j = 0; j < wireList.size(); j++)
                {
                    int k = e.getX();
                    int l = e.getY();

                    if(wireList.get(j).isSelected(k, l) && removeStatus
== true)
                    {
                        System.out.println("K: " + k + ",L: " + l );
                        System.out.println("X: " +
wireList.get(j).getStartX());
                        System.out.println("J1: " + j );
                        if(wireList.get(j).getObjectTo().getType() ==
3)
                        {
                            if(wireList.get(j).getLinkedInput() ==
1)
                            {

    wireList.get(j).getObjectTo().setInputOneUse(false);
                            }
                            else if(wireList.get(j).getLinkedInput()
== 2)
                            {
```

```
        wireList.get(j).getObjectTo().setInputTwoUse(false);
                                                }
                                        }
                                        else
                                        {

        wireList.get(j).getObjectTo().setInputOneUse(false);
                                        }


                                        wireList.remove(j);
                                        System.out.println("asd " + j);
                                        j = j - 1;
                                        removeStatus = false;

                                        repaint();
                                }
                        }
                }
        }




        public void mouseEntered(MouseEvent e)
        {

        }

        public void mouseExited(MouseEvent e)
        {

        }

        public void mousePressed(MouseEvent e)
        {
                test = true;
                pressedGate = 0;

                for(int i = 0; i < gateList.size(); i++)
                {
                        if(test)
                        {
                                x = (gateList.get(i)).getIconX();
                                y = (gateList.get(i)).getIconY();
                                width = (gateList.get(i)).getIconWidth();
                                height = (gateList.get(i)).getIconHeight();



                                if(e.getX() < x + width  && e.getX() > x  && e.getY()
    < y + height && e.getY() > y)
                                {
                                        addMouseMotionListener(myListener);
                                        test = false;
                                }
                                else
                                {
                                        pressedGate++;
                                }
                        }
                }
            }

            public void mouseReleased(MouseEvent e)
            {
                removeMouseMotionListener(myListener);
                test2 = true;
            }
        }

        public class MotionListener implements MouseMotionListener
    {
        public void mouseMoved(MouseEvent e)
        {

        }

        public void mouseDragged(MouseEvent e)
        {
            if(test2)
            {
                a = e.getX() - ((gateList.get(pressedGate)).getIconX());
                b = e.getY() - ((gateList.get(pressedGate)).getIconY());

                test2 = false;
            }

            int temp = 0;
            if(((gateList.get(pressedGate)).getIconX()) >= 100 && e.getX()- a >= 100)
            {
                int gateWidth = (gateList.get(pressedGate)).getIconWidth();

                if(((gateList.get(pressedGate)).getIconX()) + gateWidth <=
(panelWidth-100) &&
                        (e.getX() + gateWidth - a) <= (panelWidth-100))
                {
                        temp = (e.getX() - a) % 10;
                        temp = e.getX() - a - temp;
                        (gateList.get(pressedGate)).setIconX(temp);
                }
            }

            if(((gateList.get(pressedGate)).getIconY()) >= 0 && e.getY()- b >= 0)
            {
                int gateHeight = (gateList.get(pressedGate)).getIconHeight();

                if(((gateList.get(pressedGate)).getIconY()) + gateHeight <=
(panelHeight) &&
                      (e.getY() + gateHeight - b) <= (panelHeight))
                {
                        temp = (e.getY() - b) % 10;
                        temp = e.getY() - b - temp;
                        (gateList.get(pressedGate)).setIconY(temp);
                }
            }
            repaint();
        }
    }
}
```

```
/*********************************************************************************
A class that represents user interface of the circuit schematic. All operations and
user interface components such as buttons, panels are collected in SchematicGUI.
*********************************************************************************/
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

class SchematicGUI extends JPanel implements ActionListener
{
	//Properties
	private SchematicOperationGUI schGui;

	private JPanel  buttonPanel;
	private JPanel  inputOutputPanel;

	private JButton ANDButton;
	private JButton ORButton;
	private JButton NANDButton;
	private JButton NORButton;
	private JButton XORButton;
	private JButton XNORButton;
	private JButton NOTButton;
	private JButton wireButton;
	private JButton RemoveButton;
	private JButton ComputeButton;

	//YENI
	private JLabel[] inputLabels;
	private JLabel[] outputLabels;
	private JComboBox[] inputBoxes;

	private int[] schematicInput;
	private String[] outputNames;

	//Constructor
	public SchematicGUI(int inputNo, int outputNo, String[] outNames)
	{
		schGui = new SchematicOperationGUI(inputNo, outputNo, outNames);
	//	schGui.setBackground(Color.);

		buttonPanel = new JPanel(new GridLayout(10,1));
		buttonPanel.setBorder(BorderFactory.createTitledBorder("Component
List"));

		inputOutputPanel = new JPanel();
		inputOutputPanel.setBackground(Color.GRAY);

		ANDButton  = new JButton("AND Gate");
		ANDButton.addActionListener(this);

		ORButton = new JButton("OR Gate");
		ORButton.addActionListener(this);

		NANDButton  = new JButton("NAND Gate");
		NANDButton.addActionListener(this);

		NORButton = new JButton("NOR Gate");
		NORButton.addActionListener(this);

		XORButton = new JButton("XOR Gate");
```

```
XORButton.addActionListener(this);

XNORButton = new JButton("XNOR Gate");
XNORButton.addActionListener(this);

NOTButton = new JButton("NOT Gate");
NOTButton.addActionListener(this);

wireButton = new JButton("Wire");
wireButton.addActionListener(this);

RemoveButton = new JButton("Remove");
RemoveButton.addActionListener(this);

ComputeButton = new JButton("Compute");
ComputeButton.addActionListener(this);
buttonPanel.add(ANDButton);
buttonPanel.add(ORButton);
buttonPanel.add(NANDButton);
buttonPanel.add(NORButton);
buttonPanel.add(XORButton);
buttonPanel.add(XNORButton);
buttonPanel.add(NOTButton);
buttonPanel.add(wireButton);
buttonPanel.add(RemoveButton);
buttonPanel.add(ComputeButton);

String[] inputNames = new String[4];

inputNames[0] = "A";
inputNames[1] = "B";
inputNames[2] = "C";
inputNames[3] = "D";


inputLabels  = new JLabel[inputNo];
outputLabels = new JLabel[outputNo];
inputBoxes   = new JComboBox[inputNo];
for(int i = 0; i < inputNo; i++)
{
       inputLabels[i] = new JLabel();
       inputLabels[i].setText("Input " + inputNames[i] + ": ");


       inputBoxes[i] = new JComboBox();
       inputBoxes[i].addItem(0);
       inputBoxes[i].addItem(1);
       inputBoxes[i].addActionListener(this);

       inputOutputPanel.add(inputLabels[i]);
       inputOutputPanel.add(inputBoxes[i]);
}

for(int k = 0; k < outputNo; k++)
{
       outputLabels[k] = new JLabel();
       outputLabels[k].setText("Output " + outNames[k] + ": ");

       inputOutputPanel.add(outputLabels[k]);
}

setLayout( new BorderLayout());
```

```
            add(buttonPanel, BorderLayout.EAST);
            add(schGui, BorderLayout.CENTER);
            add(inputOutputPanel, BorderLayout.SOUTH);

            schematicInput = new int[4];
            for(int i = 0; i < 4; i++)
            {
                  schematicInput[i] = 0;
            }

            outputNames = new String[outputNo];
            for(int t = 0; t < outputNo; t++)
            {
                  outputNames[t] = outNames[t];
            }


      }


      //Methods
      public void actionPerformed(ActionEvent e)
      {
            if(e.getActionCommand().equals("AND Gate"))
            {
                  ImageIcon andIcon  = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\2-in-AND.png");

            //     ANDGate newGate = new ANDGate();
                  ANDGateAdapter newGate = new ANDGateAdapter();
                  newGate.setIcon(andIcon);
                  newGate.setIconX(100);
            newGate.setIconY(100);
            newGate.setIconWidth(andIcon.getIconWidth());
                  newGate.setIconHeight(andIcon.getIconHeight());
                  newGate.setInputLocations(100, 110, 100, 129);
                  newGate.setOutputLocation(150, 120);

                  schGui.addGateToList(newGate);
                  schGui.repaint();
            }
            else if(e.getActionCommand().equals("OR Gate"))
            {
                  ImageIcon orIcon = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\2-in-OR.png");
                  ORGate newGate = new ORGate();
                  newGate.setIcon(orIcon);
                  newGate.setIconX(100);
            newGate.setIconY(100);
            newGate.setIconWidth(orIcon.getIconWidth());
                  newGate.setIconHeight(orIcon.getIconHeight());
            newGate.setInputLocations(100, 110, 100, 129);
                  newGate.setOutputLocation(150, 120);

                  schGui.addGateToList(newGate);
                  schGui.repaint();

            }
            else if(e.getActionCommand().equals("NAND Gate"))
            {
                  ImageIcon nandIcon = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\2-in-NAND.png");
```

```
                        NANDGate newGate = new NANDGate();
                        newGate.setIcon(nandIcon);
                        newGate.setIconX(100);
                newGate.setIconY(100);
                newGate.setIconWidth(nandIcon.getIconWidth());
                        newGate.setIconHeight(nandIcon.getIconHeight());
                    newGate.setInputLocations(100, 110, 100, 129);
                        newGate.setOutputLocation(150, 120);

                        schGui.addGateToList(newGate);
                        schGui.repaint();

                }
                else if(e.getActionCommand().equals("NOR Gate"))
                {
                        ImageIcon norIcon = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\2-in-NOR.png");
                        NORGate newGate = new NORGate();
                        newGate.setIcon(norIcon);
                        newGate.setIconX(100);
                newGate.setIconY(100);
                newGate.setIconWidth(norIcon.getIconWidth());
                        newGate.setIconHeight(norIcon.getIconHeight());
                        newGate.setInputLocations(100, 110, 100, 129);
                        newGate.setOutputLocation(150, 120);

                        schGui.addGateToList(newGate);
                        schGui.repaint();

                }
                else if(e.getActionCommand().equals("XOR Gate"))
                {
                        ImageIcon xorIcon = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\XOR.png");
                        XORGate newGate = new XORGate();
                        newGate.setIcon(xorIcon);
                        newGate.setIconX(100);
                newGate.setIconY(100);
                newGate.setIconWidth(xorIcon.getIconWidth());
                        newGate.setIconHeight(xorIcon.getIconHeight());
                        newGate.setInputLocations(100, 110, 100, 129);
                        newGate.setOutputLocation(150, 120);

                        schGui.addGateToList(newGate);
                        schGui.repaint();
                }
                else if(e.getActionCommand().equals("XNOR Gate"))
                {
                        ImageIcon xnorIcon = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\XNOR.png");
                        XNORGate newGate = new XNORGate();
                        newGate.setIcon(xnorIcon);
                        newGate.setIconX(100);
                newGate.setIconY(100);
                newGate.setIconWidth(xnorIcon.getIconWidth());
                        newGate.setIconHeight(xnorIcon.getIconHeight());
                        newGate.setInputLocations(100, 110, 100, 129);
                        newGate.setOutputLocation(150, 120);

                        schGui.addGateToList(newGate);
                        schGui.repaint();
```

```
            }

            else if(e.getActionCommand().equals("NOT Gate"))
            {
                    ImageIcon NotIcon = new ImageIcon("C:\\Documents and
Settings\\erdinç\\Desktop\\gates\\NOT.png");
                    NOTGate newGate = new NOTGate();
                    newGate.setIcon(NotIcon);
                    newGate.setIconX(100);
            newGate.setIconY(100);
            newGate.setIconWidth(NotIcon.getIconWidth());
                    newGate.setIconHeight(NotIcon.getIconHeight());
                    newGate.setInputLocation(100, 120);
                    newGate.setOutputLocation(150, 120);

                    schGui.addGateToList(newGate);
                    schGui.repaint();

            }
            else if (e.getActionCommand().equals("Wire"))
            {
                    if((schGui.getWireList()).size() == 0 ||
(schGui.getWireList()).get(schGui.getWireList().size()-1).getStartX() != 0)
                    {
                            Wire newWire = new Wire();
                            schGui.addWireToList(newWire);
                    }

            }
            else if(e.getActionCommand().equals("Remove"))
            {
                    schGui.setRemoveStatus(true);
            }
            else if(e.getActionCommand().equals("Compute"))
            {

                    schGui.getSchematicIO().setInputValues(schematicInput);
                    schGui.computeWholeSystem(schGui.getSchematicIO());

                    for(int k = 0; k < schGui.getSchematicIO().getOutputNumber(); k++)
                    {
                            outputLabels[k].setText("Output " + outputNames[k] + ": " +
schGui.getSchematicIO().getOutputArr()[k].getOutputValue());
                            schGui.getSchematicIO().getOutputArr()[k].setOutputValue(-
1);
                    }
            }
            else
            {
                    for(int k = 0; k < schGui.getSchematicIO().getInputNumber(); k++)
                    {
                            schematicInput[k] = inputBoxes[k].getSelectedIndex();
                    }
            }


    }
}


--------------------------------------------------------------------------
/*
```

```
BBEditPanel is the interface to make modifications on Breadboard and Workstation
circuits. It intends to visualize both of them and some additional necessary components
which enable some of functionality
*/
-----------------------------------------------------------------------

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.util.ArrayList;

import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;


public class BBEditPanel extends JPanel {
	private static final long serialVersionUID = 1L;
	private static final int BREADBOARDLEVEL = 2, COMPONENTLEVEL = 1, SOCKETLEVEL =
0;
	private JLayeredPane layeredPane;
	private BuildBBControl control;
	private Breadboard breadboard;
	private JLabel breadboardLabel;
	private ArrayList<SocketLabel> socketLabels;
	private ArrayList<ChipLabel> chipLabels;
	private Workstation workstation;
	private JLabel workstationLabel;
	private ArrayList<SocketLabel> inputLabels;
	private ArrayList<SocketLabel> outputLabels;
	private ArrayList<SwitchLabel> switchLabels;
	private ArrayList<LedLabel> ledLabels;
	private SocketLabel groundLabel;
	private SocketLabel powerLabel;


	private static ImageIcon[] images =
	{
		new ImageIcon("src/breadboard.png"),	// 0
		new ImageIcon("src/socket.png"),	// 1
		new ImageIcon("src/chip.png"),			// 2
		new ImageIcon("src/workstation.png"),	// 3
		new ImageIcon("src/input.png"),			// 4
		new ImageIcon("src/output.png"),	// 5
		new ImageIcon("src/gp.png"),			// 6
		new ImageIcon("src/ledOn.png"),			// 7
		new ImageIcon("src/ledOff.png"),	// 8
		new ImageIcon("src/switchOn.png"),		// 9
		new ImageIcon("src/switchOff.png"),		// 10
	};

	// initializer

	public JLabel initializeLabel(ImageIcon icon, int positionX, int positionY){
		JLabel label = new JLabel(icon);
		label.setBounds(positionX, positionY, icon.getIconWidth(),
icon.getIconHeight());
		return label;
	}
	public void initializeSockets(){
		socketLabels = new ArrayList<SocketLabel>();
		for(int i = 0; i < Breadboard.BREADBOARD_ROWNUMBER / 2; i++){
			for (int j = 0; j < Breadboard.BREADBOARD_COLNUMBER; j++) {
				SocketLabel adder = new SocketLabel(images[1],
						j * Socket.GAP + Socket.SIDE,
						i * Socket.GAP + Socket.SIDE,
						this.control);
				socketLabels.add(adder);
				adder.setSocket(breadboard.getSocket(new Point(j, i)));
				this.layeredPane.add(adder, SOCKETLEVEL);
			}
		}
		for(int i = Breadboard.BREADBOARD_ROWNUMBER / 2; i <
Breadboard.BREADBOARD_ROWNUMBER; i++){
			for (int j = 0; j < Breadboard.BREADBOARD_COLNUMBER; j++) {
				SocketLabel adder = new SocketLabel(images[1],
						j * Socket.GAP + Socket.SIDE,
						(i + 1) * Socket.GAP + Socket.SIDE,
						this.control);
				socketLabels.add(adder);
				adder.setSocket(breadboard.getSocket(new Point(j, i)));
				this.layeredPane.add(adder, SOCKETLEVEL);
			}
		}
	}

	public void initializeInputAndSwitches(){
		inputLabels = new ArrayList<SocketLabel>();
		switchLabels = new ArrayList<SwitchLabel>();
		for(int i = workstation.getOutputs().size(); i <
workstation.getOutputs().size() + workstation.getInputs().size(); i++){
			SocketLabel adder = new SocketLabel(images[4],
					i * Socket.GAP + Socket.SIDE,
					workstation.getPosition().y + Socket.SIDE,
					this.control);
			inputLabels.add(adder);
			adder.setSocket(breadboard.getSocket(new Point(i,14)));
			this.layeredPane.add(adder, COMPONENTLEVEL);
			SwitchLabel switchAdder = new SwitchLabel(images[10],
					i * Socket.GAP + Socket.SIDE,
					workstation.getPosition().y + Socket.SIDE +
Socket.GAP,
					this.control);
			switchLabels.add(switchAdder);
			switchAdder.setInputSwitch(workstation.getSwitch(new Point(i,14)));
			this.layeredPane.add(switchAdder, COMPONENTLEVEL);
		}
	}

	public void initializeOutputAndLeds(){
		outputLabels = new ArrayList<SocketLabel>();
		ledLabels = new ArrayList<LedLabel>();
		for(int i = 0; i < workstation.getOutputs().size(); i++){
			SocketLabel adder = new SocketLabel(images[5],
					i * Socket.GAP + Socket.SIDE,
					workstation.getPosition().y + Socket.SIDE,
					this.control);
			outputLabels.add(adder);
			adder.setSocket(breadboard.getSocket(new Point(i,14)));
			this.layeredPane.add(adder, COMPONENTLEVEL);
			LedLabel ledAdder = new LedLabel(images[8],
                                 i * Socket.GAP + Socket.SIDE,
                                 workstation.getPosition().y + Socket.SIDE +
Socket.GAP,
                                 this.control);
                    ledLabels.add(ledAdder);
                    ledAdder.setLed(workstation.getLed(new Point(i,14)));
                    this.layeredPane.add(ledAdder, COMPONENTLEVEL);
             }
      }

      public void addComponents(){

      }

      // constructor

      public BBEditPanel(Breadboard breadboard, Workstation workstation,
BuildBBControl control){
             super();
             this.setBackground(Color.black);
             this.setLayout(null);
             this.layeredPane = new JLayeredPane();
             this.layeredPane.setBackground(Color.darkGray);
             this.layeredPane.setLayout(null);
             this.layeredPane.setOpaque(true);
             this.layeredPane.setBounds(0, 0, 610, 410);
             this.control = control;
             this.breadboard = breadboard;
             this.breadboardLabel = initializeLabel(images[0],
breadboard.getPosition().x, breadboard.getPosition().y);
             this.layeredPane.add(breadboardLabel,BREADBOARDLEVEL);
             this.workstation = workstation;
             this.workstation.setPosition(new
Point(0,breadboardLabel.getIcon().getIconHeight()));
             this.workstationLabel = initializeLabel(images[3],
workstation.getPosition().x, workstation.getPosition().y);
             this.workstation.setPosition(new Point(0,breadboardLabel.getHeight()));
             this.layeredPane.add(workstationLabel,BREADBOARDLEVEL);
             this.initializeSockets();
             this.initializeInputAndSwitches();
             this.initializeOutputAndLeds();
             this.add(layeredPane);
      }

      /*this.groundLabel = new SocketLabel(images[6],
                    workstation.getPosition().x +
                    (workstation.getInputs().size() + workstation.getOutputs().size())
* Socket.GAP +
                    Socket.SIDE,
                    workstation.getPosition().y +
                    Socket.SIDE);*/

      /*this.powerLabel = new SocketLabel(images[6],
      workstation.getPosition().x +
      groundLabel.getLocation().x + Socket.GAP,
      workstation.getPosition().y +
      Socket.SIDE);*/

      // methods

      @Override
      public void paint(Graphics g){
```

```
		super.paint(g);
		//Graphics2D g2 = (Graphics2D) g;
	}

	@Override
	public void update(Graphics g){
		paint(g);
	}

	// setter & getter

	public void setControl(BuildBBControl control) {
		this.control = control;
	}

	public BuildBBControl getControl() {
		return control;
	}

	public void setBreadboard(Breadboard breadboard) {
		this.breadboard = breadboard;
	}

	public Breadboard getBreadboard() {
		return breadboard;
	}

	public void setWorkstation(Workstation workstation) {
		this.workstation = workstation;
	}

	public Workstation getWorkstation() {
		return workstation;
	}

	public JLabel getBreadboardLabel() {
		return breadboardLabel;
	}

	public void setBreadboardLabel(JLabel breadboardLabel) {
		this.breadboardLabel = breadboardLabel;
	}

	public ArrayList<SocketLabel> getSocketLabels() {
		return socketLabels;
	}

	public void setSocketLabels(ArrayList<SocketLabel> socketLabels) {
		this.socketLabels = socketLabels;
	}

	public ArrayList<ChipLabel> getChipLabels() {
		return chipLabels;
	}

	public void setChipLabels(ArrayList<ChipLabel> chipLabels) {
		this.chipLabels = chipLabels;
	}

	public JLabel getWorkstationLabel() {
		return workstationLabel;
	}
	public void setWorkstationLabel(JLabel workstationLabel) {
		this.workstationLabel = workstationLabel;
	}

	public ArrayList<SocketLabel> getInputLabels() {
		return inputLabels;
	}

	public void setInputLabels(ArrayList<SocketLabel> inputLabels) {
		this.inputLabels = inputLabels;
	}

	public ArrayList<SocketLabel> getOutputLabels() {
		return outputLabels;
	}

	public void setOutputLabels(ArrayList<SocketLabel> outputLabels) {
		this.outputLabels = outputLabels;
	}

	public ArrayList<SwitchLabel> getSwitchLabels() {
		return switchLabels;
	}

	public void setSwitchLabels(ArrayList<SwitchLabel> switchLabels) {
		this.switchLabels = switchLabels;
	}

	public ArrayList<LedLabel> getLedLabels() {
		return ledLabels;
	}

	public void setLedLabels(ArrayList<LedLabel> ledLabels) {
		this.ledLabels = ledLabels;
	}

	public SocketLabel getGroundLabel() {
		return groundLabel;
	}

	public void setGroundLabel(SocketLabel groundLabel) {
		this.groundLabel = groundLabel;
	}

	public SocketLabel getPowerLabel() {
		return powerLabel;
	}

	public void setPowerLabel(SocketLabel powerLabel) {
		this.powerLabel = powerLabel;
	}

	public static void setImages(ImageIcon[] images) {
		BBEditPanel.images = images;
	}

	public static ImageIcon[] getImages() {
		return images;
	}
}
------------------------------------------------------------------------
```

```
/*
Breadboard entity. Holds sockets, chips and wires.
*/
----------------------------------------------------------------------

import java.awt.Point;
import java.util.ArrayList;
import java.util.Iterator;


public class Breadboard implements Locatable {
	public static final int BREADBOARD_ROWNUMBER = 14,
					    BREADBOARD_COLNUMBER = 30,
					    LAST_CHIP_POSITION = 23;
	private Point position;
	private ArrayList<Socket> sockets;
	private ArrayList<Chip> chips;
	private ArrayList<Wire> wires;

	// initializers

	private void initializeSocketsWithPosition(int numOfSocket){
		this.sockets = new ArrayList<Socket>();
		for(int i = 0; i < numOfSocket; i++){
			this.sockets.add(new Socket(i % BREADBOARD_COLNUMBER, i /
BREADBOARD_COLNUMBER));
		}
	}

	// constructors

	public Breadboard(){
		this(0,0);
	}

	public Breadboard(int positionX, int positionY) {
		this.setPosition(new Point(positionX, positionY));
		this.initializeSocketsWithPosition(BREADBOARD_COLNUMBER *
BREADBOARD_ROWNUMBER);
		this.chips = new ArrayList<Chip>();
		this.wires = new ArrayList<Wire>();
	}

	// methods

	public boolean addChip(Socket[] sockets){
		if(sockets == null || sockets.length != BREADBOARD_ROWNUMBER){
			System.out.println("sockets given are not enough to put chip on or
no sockets are given");
			return false;
		}
		for(int i = 0; i < sockets.length; i++){
			if(sockets[i] == null){
				System.out.println("one of the sockets is null");
				return false;
			}
		}
		for(int i = 0; i < sockets.length; i++){
			if(sockets[i].getState() == true){
				System.out.println("one of the sockets given is already
connected");
				return false;
```

```
                        }
                }
                Chip chip = new ANDChip(sockets);
                boolean additionSuccessful = this.chips.add(chip);
                if(additionSuccessful){
                        setSocketsForChip(sockets[0].getPosition().x, chip);
                }
                return additionSuccessful;
        }
        public boolean removeChip(int positionX){
                if(positionX < 0 || LAST_CHIP_POSITION < positionX){
                        System.out.println("position given are out of bounds of 
breadboard");
                        return false;
                }
                boolean removalSuccessful = false;
                Chip c = null;
                for(int i = 0; i < this.getChips().size(); i++){
                        c = this.getChips().get(i);
                        if(c.getSockets()[0].getPosition().x <= positionX &&
                                        positionX < c.getSockets()[0].getPosition().x + 
Chip.CHIP_WIDTH){
                                System.out.println("found the chip to be removed");
                                removalSuccessful = this.chips.remove(c);
                                break;
                        }
                }
                if(removalSuccessful && !c.equals(null)){
                        resetSocketsForChip(c.getSockets()[0].getPosition().x);
                }
                return removalSuccessful;
        }

        public boolean addWire(Wire wire, String whichEnd, boolean firstEnd){
                boolean additionSuccessful = false;
                if (wire == null || wire.getSockets().length != 2) {
                        System.out.println("given wire has some problems");
                        return false;
                }
                if (wire.getSockets()[0] == null || wire.getSockets()[1] == null) {
                        System.out.println("one end of the wire is null");
                        return false;
                }
                if (wire.getSockets()[0].getPosition().x == 
wire.getSockets()[1].getPosition().x &&
                                wire.getSockets()[0].getPosition().y == 
wire.getSockets()[1].getPosition().y){
                        System.out.println("wire from one socket to same socket is not 
allowed.");
                        return false;
                }
                if (whichEnd.equals("begin") && wire.getSockets()[0].getPosition().y < 
BREADBOARD_ROWNUMBER) {
                        if (wire.getSockets()[0].getState() == true ) {
                                System.out.println("socket on one of the end of given wire 
is already connected");
                                return false;
                        }
                        if(firstEnd){
                                additionSuccessful = this.wires.add(wire);
                        }
                        else{
```

```
                        additionSuccessful = true;
                    }
                    if (additionSuccessful) {
                        connectSocketTo(wire.getSockets()[0].getPosition(), wire);
                    }
            }
            if(whichEnd.equals("end") && wire.getSockets()[1].getPosition().y <
BREADBOARD_ROWNUMBER){
                    if (wire.getSockets()[1].getState() == true ) {
                        System.out.println("socket on one of the end of given wire
is already connected");
                        return false;
                    }
                    if(firstEnd){
                        additionSuccessful = this.wires.add(wire);
                    }
                    else{
                        additionSuccessful = true;
                    }
                    if(additionSuccessful){
                        connectSocketTo(wire.getSockets()[1].getPosition(), wire);
                    }
            }
            return additionSuccessful;
    }

    public boolean removeWire(Wire wire, String whichEnd){
            boolean removalSuccessful = false;
            if (wire == null || wire.getSockets().length != 2) {
                    System.out.println("given wire has some problems");
                    return false;
            }
            if (wire.getSockets()[0] == null || wire.getSockets()[1] == null) {
                    System.out.println("one end of the wire is null");
                    return false;
            }
            if (whichEnd.equals("begin")) {
                    removalSuccessful = this.wires.remove(wire);
                    if (removalSuccessful) {
                        disconnectSocket(new
Point(wire.getSockets()[0].getPosition().x, wire.getSockets()[0].getPosition().y));
                    }
            }
            if(whichEnd.equals("end")){
                    removalSuccessful = this.wires.remove(wire);
                    if(removalSuccessful){
                        disconnectSocket(new
Point(wire.getSockets()[1].getPosition().x, wire.getSockets()[1].getPosition().y));
                    }
            }
            return removalSuccessful;
    }

    public boolean checkSocket(Point socketPosition){
            Iterator<Socket> iterator = this.sockets.iterator();
            while(iterator.hasNext()){
                    Socket s = iterator.next();
                    if(s.getPosition().x == socketPosition.x && s.getPosition().y ==
socketPosition.y){
                        return s.getState();
                    }
            }
```

```
			System.out.println("no socket is found at position specified");
			return true;
		}

		public boolean connectSocketTo(Point socketPosition, CircuitComponent 
component){
			Socket s = getSocket(socketPosition);
			if(s != null){
				s.setConnectedComponent(component);
				return true;
			}
			return false;
		}

		public boolean disconnectSocket(Point socketPosition){
			Socket s = getSocket(socketPosition);
			if(s != null){
				s.setConnectedComponent(null);
				return true;
			}
			return false;
		}

		public Socket getSocket(Point socketPosition){
			Iterator<Socket> iterator = this.sockets.iterator();
			while(iterator.hasNext()){
				Socket s = iterator.next();
				if(s.getPosition().x == socketPosition.x && s.getPosition().y == 
socketPosition.y){
					return s;
				}
			}
			return null;
		}

		public Socket[] getSocketsForChip(int positionX){
			if(positionX < 0 || LAST_CHIP_POSITION < positionX){
				System.out.println("out of breadboard bounds, cannot retrieve 
sockets for chip");
				return null;
			}
			Socket[] returnArray = new Socket[Chip.NUM_OF_SOCKETS];
			Point fetcher = new Point(positionX,0);
			for(int i = 0; i < Chip.CHIP_WIDTH; i++){
				fetcher.y = 6;
				returnArray[i] = getSocket(fetcher);
				fetcher.y = 7;
				returnArray[i + Chip.CHIP_WIDTH] = getSocket(fetcher);
				fetcher.x++;
			}
			return returnArray;
		}

		public void setSocketsForChip(int positionX, Chip chip){
			if(positionX < 0 || LAST_CHIP_POSITION < positionX){
				System.out.println("out of breadboard bounds, cannot set sockets 
for chip");
				return;
			}
			Point fetcher = new Point(positionX,0);
			for(int i = 0; i < Chip.CHIP_WIDTH; i++){
				fetcher.y = 6;
```

```
                    connectSocketTo(fetcher, chip);
                    fetcher.y = 7;
                    connectSocketTo(fetcher, chip);
                    fetcher.x++;
                }
        }

        public void resetSocketsForChip(int positionX){
                if(positionX < 0 || LAST_CHIP_POSITION < positionX){
                        System.out.println("out of breadboard bounds, cannot set sockets
for chip");
                        return;
                }
                Point fetcher = new Point(positionX,0);
                for(int i = 0; i < Chip.CHIP_WIDTH; i++){
                        fetcher.y = 6;
                        disconnectSocket(fetcher);
                        fetcher.y = 7;
                        disconnectSocket(fetcher);
                        fetcher.x++;
                }
        }

        // setter & getter

        @Override
        public Point getPosition() {
                return position;
        }

        @Override
        public void setPosition(Point point) {
                this.position = point;
        }

        public ArrayList<Socket> getSockets() {
                return sockets;
        }

        public void setSockets(ArrayList<Socket> sockets) {
                this.sockets = sockets;
        }

        public ArrayList<Chip> getChips() {
                return chips;
        }

        public void setChips(ArrayList<Chip> chips) {
                this.chips = chips;
        }

        public ArrayList<Wire> getWires() {
                return wires;
        }

        public void setWires(ArrayList<Wire> wires) {
                this.wires = wires;
        }
}
------------------------------------------------------------------------
/*
The control system over the Breadboard entity, Workstation entity and user interfaces.
```

```
*/
------------------------------------------------------------------------

import java.awt.Point;

import javax.swing.JFrame;


public class BuildBBControl {
	private Breadboard breadboard;
	private Workstation workstation;
	private BBEditPanel editPanel;

	public static void main(String[] args) {
		BuildBBControl main = new BuildBBControl();
		main.breadboard = new Breadboard();
		main.workstation = new Workstation();
		main.editPanel = new BBEditPanel(main.breadboard, main.workstation,
main);
		// test begins here
		JFrame frame = new JFrame();
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.add(main.editPanel);
		frame.setBounds(0, 0, 610 + 16, 310 + 100 + 38);
		frame.setVisible(true);
		//System.out.println(main.addWire(0, 0, 8, 14));

		// test ends here
		main.breadboard = null;
	}

	public boolean addWire(int beginX, int beginY, int endX, int endY){
		Socket[] sockets = new Socket[2];
		if(beginY < 14){
			sockets[0] = this.breadboard.getSocket(new Point(beginX,beginY));
		}
		if(beginY == 14){
			sockets[0] = this.workstation.getSocket(new Point(beginX,beginY));
		}
		if(endY < 14){
			sockets[1] = this.breadboard.getSocket(new Point(endX,endY));
		}
		if(endY == 14){
			sockets[1] = this.workstation.getSocket(new Point(endX,endY));
		}
		Wire wire = new Wire(sockets);
		boolean additionalSuccessful = false;
		if(wire.getSockets()[0].getPosition().y < 14){
			if(wire.getSockets()[1].getPosition().y < 14){
				additionalSuccessful = this.breadboard.addWire(wire,
"begin", true);
				if(!additionalSuccessful){
					return additionalSuccessful;
				}
				additionalSuccessful = this.breadboard.addWire(wire, "end",
false);
				if(!additionalSuccessful){
					this.breadboard.removeWire(wire, "begin");
					return additionalSuccessful;
				}
			}
			if(wire.getSockets()[1].getPosition().y == 14){
```

```
                        additionalSuccessful = this.breadboard.addWire(wire,
"begin", true);
                        if(!additionalSuccessful){
                              return additionalSuccessful;
                        }
                        additionalSuccessful = this.workstation.addWire(wire, "end",
true);
                        if(!additionalSuccessful){
                              this.breadboard.removeWire(wire, "begin");
                              return additionalSuccessful;
                        }
                  }
            }
            if(wire.getSockets()[0].getPosition().y == 14){
                  if(wire.getSockets()[1].getPosition().y < 14){
                        additionalSuccessful = this.workstation.addWire(wire,
"begin", true);
                        if(!additionalSuccessful){
                              return additionalSuccessful;
                        }
                        additionalSuccessful = this.breadboard.addWire(wire, "end",
true);
                        if(!additionalSuccessful){
                              this.breadboard.removeWire(wire, "begin");
                              return additionalSuccessful;
                        }
                  }
                  if(wire.getSockets()[1].getPosition().y == 14){
                        additionalSuccessful = this.workstation.addWire(wire,
"begin", true);
                        if(!additionalSuccessful){
                              return additionalSuccessful;
                        }
                        additionalSuccessful = this.workstation.addWire(wire, "end",
false);
                        if(!additionalSuccessful){
                              this.breadboard.removeWire(wire, "begin");
                              return additionalSuccessful;
                        }
                  }
            }
            return additionalSuccessful;
      }

      public boolean removeWire(int positionX, int positionY){
            Socket socket;
            boolean removalSuccessful = false;
            if(positionY < 14){
                  socket = this.breadboard.getSocket(new Point(positionX,positionY));
                  if(socket.getConnectedComponent() instanceof Wire){
                        Wire wire = (Wire) socket.getConnectedComponent();
                        if(wire.getSockets()[0].getPosition().y < 14){
                              removalSuccessful = this.breadboard.removeWire(wire,
"begin");
                              if(removalSuccessful &&
wire.getSockets()[1].getPosition().y < 14){

      this.breadboard.disconnectSocket(wire.getSockets()[1].getPosition());
                              }
                              else if(removalSuccessful &&
wire.getSockets()[1].getPosition().y == 14){
```

```
                                        removalSuccessful = 
this.workstation.removeWire(wire, "end");
                                    }
                            }
                            else if(wire.getSockets()[0].getPosition().y == 14){
                                    removalSuccessful = this.workstation.removeWire(wire, 
"begin");
                                    if(removalSuccessful && 
wire.getSockets()[1].getPosition().y < 14){
                                            removalSuccessful = 
this.breadboard.removeWire(wire, "end");
                                    }
                                    else if(removalSuccessful = 
wire.getSockets()[1].getPosition().y == 14){

        this.workstation.disconnectSocket(wire.getSockets()[1].getPosition());
                                    }
                            }
                    }
            }
            else if(positionY == 14){
                    socket = this.workstation.getSocket(new 
Point(positionX,positionY));
                    if(socket.getConnectedComponent() instanceof Wire){
                            Wire wire = (Wire) socket.getConnectedComponent();
                            if(wire.getSockets()[0].getPosition().y < 14){
                                    removalSuccessful = this.breadboard.removeWire(wire, 
"begin");
                                    if(removalSuccessful && 
wire.getSockets()[1].getPosition().y < 14){

        this.breadboard.disconnectSocket(wire.getSockets()[1].getPosition());
                                    }
                                    else if(removalSuccessful && 
wire.getSockets()[1].getPosition().y == 14){
                                            removalSuccessful = 
this.workstation.removeWire(wire, "end");
                                    }
                            }
                            else if(wire.getSockets()[0].getPosition().y == 14){
                                    removalSuccessful = this.workstation.removeWire(wire, 
"begin");
                                    if(removalSuccessful && 
wire.getSockets()[1].getPosition().y < 14){
                                            removalSuccessful = 
this.breadboard.removeWire(wire, "end");
                                    }
                                    else if(removalSuccessful = 
wire.getSockets()[1].getPosition().y == 14){

        this.workstation.disconnectSocket(wire.getSockets()[1].getPosition());
                                    }
                            }
                    }
            }
            else{
                    System.out.println("given position is not available");
                    return false;
            }
            return removalSuccessful;
    }
```

```
	public Breadboard getBreadboard() {
		return breadboard;
	}


	public void setBreadboard(Breadboard breadboard) {
		this.breadboard = breadboard;
	}


	public Workstation getWorkstation() {
		return workstation;
	}


	public void setWorkstation(Workstation workstation) {
		this.workstation = workstation;
	}


	public BBEditPanel getEditPanel() {
		return editPanel;
	}


	public void setEditPanel(BBEditPanel editPanel) {
		this.editPanel = editPanel;
	}
}
```

```
--------------------------------------------------------------------------
/*
Abstraction of all chip entities. Its child classes will be real chip entity classes. A
chip contains several gates
*/
--------------------------------------------------------------------------
import java.util.ArrayList;


public abstract class Chip extends CircuitComponent {
	public static final int CHIP_WIDTH = 7,
	                                         NUM_OF_SOCKETS = 14;

	protected ArrayList<Gate> gates;

	public Chip(){
		this.sockets = new Socket[14];
	}

	public ArrayList<Gate> getGates() {
		return gates;
	}

	public void setGates(ArrayList<Gate> gates) {
		this.gates = gates;
	}
}
--------------------------------------------------------------------------
/*
Graphical representation class of a Chip on the Breadboard. Each instance is associated
with a chip entity object.
```

```
*/
--------------------------------------------------------------------------
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.ImageIcon;
import javax.swing.JLabel;


public class ChipLabel extends JLabel implements MouseListener{
	private static final long serialVersionUID = 1L;
	private Chip chip;
	private BuildBBControl control;

	public ChipLabel(ImageIcon icon, int positionX, int positionY, BuildBBControl
control){
		super(icon);
		this.setBounds(positionX, positionY, icon.getIconWidth(),
icon.getIconHeight());
		this.addMouseListener(this);
		this.control = control;
	}

	public void setChip(Chip chip) {
		this.chip = chip;
	}

	public Chip getChip() {
		return chip;
	}

	public void setControl(BuildBBControl control) {
		this.control = control;
	}

	public BuildBBControl getControl() {
		return control;
	}

	@Override
	public void mouseClicked(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseEntered(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseExited(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mousePressed(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}
	@Override
	public void mouseReleased(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}
}


-------------------------------------------------------------------------
/*
Abstraction of the components which can be connected to Breadboard - which are Chip and
Wire.
*/
-------------------------------------------------------------------------
public abstract class CircuitComponent {
	protected Socket[] sockets;

	public Socket[] getSockets() {
		return sockets;
	}

	public void setSockets(Socket[] sockets) {
		this.sockets = sockets;
	}
}
-------------------------------------------------------------------------
/*
Represents an output Led on the Workstation. It can be turned on or off.
*/
-------------------------------------------------------------------------
import java.awt.Point;


public class Led extends Switchable implements Locatable {
	private Point position;

	public Led(){
		this(0,0);
	}

	public Led(int positionX, int positionY){
		super();
		this.setPosition(new Point(positionX, positionY));
	}

	@Override
	public Point getPosition() {
		return this.position;
	}

	@Override
	public void setPosition(Point point) {
		this.position = point;
	}

}

-------------------------------------------------------------------------
/*
Graphical representation class of a Led on the Workstation. Each instance is associated
with a led entity object.
```

```
*/
--------------------------------------------------------------------------
import javax.swing.ImageIcon;
import javax.swing.JLabel;


public class LedLabel extends JLabel{
	private static final long serialVersionUID = 1L;
	private Led led;
	private BuildBBControl control;

	public LedLabel(ImageIcon icon, int positionX, int positionY, BuildBBControl
control){
		super(icon);
		this.setBounds(positionX, positionY, icon.getIconWidth(),
icon.getIconHeight());
		this.control = control;
	}

	public void setLed(Led led) {
		this.led = led;
	}

	public Led getLed() {
		return led;
	}

	public void setControl(BuildBBControl control) {
		this.control = control;
	}

	public BuildBBControl getControl() {
		return control;
	}
}

--------------------------------------------------------------------------
/*
Interfaces for all locatable objects, which means they have a position property.
*/
--------------------------------------------------------------------------
import java.awt.Point;


public interface Locatable {

	public Point getPosition();
	public void setPosition(Point point);
}

--------------------------------------------------------------------------
/*
Represents Socket entity on both breadboard and workstation. It can be connected to
either chip or wire.
*/
--------------------------------------------------------------------------
import java.awt.Point;


public class Socket extends Switchable implements Locatable {
	public static final int GAP = 20,
						SIDE = 10;
	private Point position;
	private CircuitComponent connectedComponent;

	public Socket(){
		this(0,0);
	}

	public Socket(int positionX, int positionY) {
		super();
		this.setPosition(new Point(positionX, positionY));
		this.setConnectedComponent(null);
	}

	@Override
	public Point getPosition() {
		return this.position;
	}

	@Override
	public void setPosition(Point point) {
		this.position = point;
	}

	public CircuitComponent getConnectedComponent() {
		return connectedComponent;
	}

	public void setConnectedComponent(CircuitComponent connectedComponent) {
		if(connectedComponent == null){
			this.state = false;
			this.connectedComponent = null;
		}
		else{
			this.state = true;
			this.connectedComponent = connectedComponent;
		}
	}

}
-------------------------------------------------------------------------
/*
Graphical representation of Socket. Each instance is associated with a socket object.
*/
-------------------------------------------------------------------------
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.ImageIcon;
import javax.swing.JLabel;


public class SocketLabel extends JLabel implements MouseListener{
	private static final long serialVersionUID = 1L;
	private Socket socket;
	private BuildBBControl control;

	public SocketLabel(ImageIcon icon, int positionX, int positionY, BuildBBControl
control) {
		super(icon);
		this.setBounds(positionX, positionY, icon.getIconWidth(),
icon.getIconHeight());
		this.addMouseListener(this);
		this.setControl(control);
	}

	public void setSocket(Socket socket) {
		this.socket = socket;
	}

	public Socket getSocket() {
		return socket;
	}

	public void setControl(BuildBBControl control) {
		this.control = control;
	}


	public BuildBBControl getControl() {
		return control;
	}


	@Override
	public void mouseClicked(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseEntered(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseExited(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mousePressed(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseReleased(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}
}

-------------------------------------------------------------------------
/*
Switch entity on the workstation. It has a state which sets or resets the input value
associated with itself
*/
-------------------------------------------------------------------------
import java.awt.Point;


public class Switch extends Switchable implements Locatable {
	private Point position;

	public Switch() {
		this(0,0);
	}

	public Switch(int positionX, int positionY){
		super();
		this.setPosition(new Point(positionX, positionY));
	}

	@Override
	public Point getPosition() {
		return this.position;
	}

	@Override
	public void setPosition(Point point) {
		this.position = point;
	}

}

----------------------------------------------------------------------------
/*
Abstraction of all object which has two option state: true and false. Includes switch,
led and socket.
*/
----------------------------------------------------------------------------

public abstract class Switchable {
	protected boolean state;

	public Switchable(){
		this.setState(false);
	}

	public boolean getState(){
		return this.state;
	}
	public void setState(boolean state){
		this.state = state;
	}
}

----------------------------------------------------------------------------
/*
Graphical representation of a switch. Each instance is associated with real switch
entity switch.
*/
----------------------------------------------------------------------------
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

import javax.swing.ImageIcon;
import javax.swing.JLabel;


public class SwitchLabel extends JLabel implements MouseListener{
	private static final long serialVersionUID = 1L;
	private Switch inputSwitch;
	private BuildBBControl control;
	public SwitchLabel(ImageIcon icon, int positionX, int positionY, BuildBBControl 
control) {
		super(icon);
		this.setBounds(positionX, positionY, icon.getIconWidth(), 
icon.getIconHeight());
		this.addMouseListener(this);
		this.setControl(control);
	}

	public void setInputSwitch(Switch inputSwitch) {
		this.inputSwitch = inputSwitch;
	}

	public Switch getInputSwitch() {
		return inputSwitch;
	}

	public void setControl(BuildBBControl control) {
		this.control = control;
	}

	public BuildBBControl getControl() {
		return control;
	}

	@Override
	public void mouseClicked(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseEntered(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseExited(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mousePressed(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseReleased(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

}
-------------------------------------------------------------------------
/*
Wire entity object. It has two ending points which can be connected to either 
breadboard or workstation
*/
```

```
--------------------------------------------------------------------------
import java.awt.Color;


public class BBWire extends CircuitComponent {
	private Color color;

	public BBWire(){
		super();
		this.setColor(generateColor());
	}

	public BBWire(Socket beginSocket, Socket endSocket){
		this();
		this.sockets = new Socket[2];
		this.getSockets()[0] = beginSocket;
		this.getSockets()[1] = endSocket;
	}

	public BBWire(Socket[] sockets){
		this();
		this.sockets = new Socket[2];
		this.setSockets(sockets);
	}

	public Color getColor() {
		return color;
	}

	public void setColor(Color color) {
		this.color = color;
	}

	public static Color generateColor(){
		int r = (int) Math.floor(Math.random() * 255);
		int g = (int) Math.floor(Math.random() * 255);
		int b = (int) Math.floor(Math.random() * 255);
		return new Color(r,g,b,80);
	}
}

--------------------------------------------------------------------------
/*
Workstation entity object. It contains socket for input, output, ground and power,
switches and leds.
*/
--------------------------------------------------------------------------
import java.awt.Point;
import java.util.ArrayList;
import java.util.Iterator;


public class Workstation implements Locatable {
	private Point position;
	private ArrayList<Wire> wires;
	private ArrayList<Socket> inputs;
	private ArrayList<Switch> switches;
	private ArrayList<Socket> outputs;
	private ArrayList<Led> leds;
	private Socket ground;
	private Socket power;
```

```
        // initializers

        public void initializeIO(){
                this.initializeIO(4,8);
        }

        public void initializeIO(int numOfInputs, int numOfOutputs){
                this.inputs = new ArrayList<Socket>();
                this.switches = new ArrayList<Switch>();
                this.outputs = new ArrayList<Socket>();
                this.leds = new ArrayList<Led>();
                for(int i = 0; i < numOfOutputs; i++){
                        this.outputs.add(new Socket(i, 14));
                        this.leds.add(new Led(i,15));
                }
                for(int j = numOfOutputs; j < numOfInputs + numOfOutputs; j++){
                        this.inputs.add(new Socket(j, 14));
                        this.switches.add(new Switch(j, 15));
                }
        }

        // constructors

        public Workstation(){
                this(0,0);
        }

        public Workstation(int positionX, int positionY){
                this.setPosition(new Point(positionX, positionY));
                this.wires = new ArrayList<Wire>();
                this.initializeIO();
                this.ground = new Socket(12, 14);
                this.power = new Socket(13, 14);
        }

        public Workstation(int positionX, int positionY, int numOfInput, int
numOfOutput){
                this.setPosition(new Point(positionX, positionY));
                this.wires = new ArrayList<Wire>();
                this.initializeIO(numOfInput, numOfOutput);
                this.ground = new Socket(numOfInput + numOfOutput, 14);
                this.power = new Socket(numOfInput + numOfOutput + 1, 14);
        }

        // methods

        public boolean addWire(Wire wire, String whichEnd, boolean firstEnd){
                boolean additionSuccessful = false;
                if (wire == null || wire.getSockets().length != 2) {
                        System.out.println("given wire has some problems");
                        return false;
                }
                if (wire.getSockets()[0] == null || wire.getSockets()[1] == null) {
                        System.out.println("one end of the wire is null");
                        return false;
                }
                if (wire.getSockets()[0].getPosition().x ==
wire.getSockets()[1].getPosition().x &&
                                wire.getSockets()[0].getPosition().y ==
wire.getSockets()[1].getPosition().y){
                        System.out.println("wire from one socket to same socket is not
allowed.");
```

```
                    return false;
            }
            if (whichEnd.equals("begin") && wire.getSockets()[0].getPosition().y ==
14) {
                    if (wire.getSockets()[0].getState() == true ) {
                          System.out.println("socket on one of the end of given wire
is already connected");
                          return false;
                    }
                    if(firstEnd){
                          additionSuccessful = this.wires.add(wire);
                    }
                    else{
                          additionSuccessful = true;
                    }
                    if (additionSuccessful) {
                          connectSocketTo(wire.getSockets()[0].getPosition(), wire);
                    }
            }
            if(whichEnd.equals("end") && wire.getSockets()[1].getPosition().y == 14){
                    if (wire.getSockets()[1].getState() == true ) {
                          System.out.println("socket on one of the end of given wire
is already connected");
                          return false;
                    }
                    if(firstEnd){
                          additionSuccessful = this.wires.add(wire);
                    }
                    else{
                          additionSuccessful = true;
                    }
                    if(additionSuccessful){
                          connectSocketTo(wire.getSockets()[1].getPosition(), wire);
                    }
            }
            return additionSuccessful;
      }

      public boolean removeWire(Wire wire, String whichEnd){
            boolean removalSuccessful = false;
            if (wire == null || wire.getSockets().length != 2) {
                  System.out.println("given wire has some problems");
                  return false;
            }
            if (wire.getSockets()[0] == null || wire.getSockets()[1] == null) {
                  System.out.println("one end of the wire is null");
                  return false;
            }
            if (whichEnd.equals("begin")) {
                  removalSuccessful = this.wires.remove(wire);
                  if (removalSuccessful) {
                        disconnectSocket(new
Point(wire.getSockets()[0].getPosition().x, wire.getSockets()[0].getPosition().y));
                  }
            }
            if(whichEnd.equals("end")){
                  removalSuccessful = this.wires.remove(wire);
                  if(removalSuccessful){
                        disconnectSocket(new
Point(wire.getSockets()[1].getPosition().x, wire.getSockets()[1].getPosition().y));
                  }
            }
```

```
		return removalSuccessful;
	}

	public boolean connectSocketTo(Point socketPosition, CircuitComponent 
component){
		Socket s = getSocket(socketPosition);
		if(s != null){
			s.setConnectedComponent(component);
			return true;
		}
		return false;
	}

	public boolean disconnectSocket(Point socketPosition){
		Socket s = getSocket(socketPosition);
		if(s != null){
			s.setConnectedComponent(null);
			return true;
		}
		return false;
	}

	public Socket getSocket(Point socketPosition){
		Iterator<Socket> iterator = this.inputs.iterator();
		while(iterator.hasNext()){
			Socket s = iterator.next();
			if(s.getPosition().x == socketPosition.x && s.getPosition().y == 
socketPosition.y){
				return s;
			}
		}
		iterator = this.outputs.iterator();
		while(iterator.hasNext()){
			Socket s = iterator.next();
			if(s.getPosition().x == socketPosition.x && s.getPosition().y == 
socketPosition.y){
				return s;
			}
		}
		Socket s = this.ground;
		if(s.getPosition().x == socketPosition.x && s.getPosition().y == 
socketPosition.y){
			return s;
		}
		s = this.power;
		if(s.getPosition().x == socketPosition.x && s.getPosition().y == 
socketPosition.y){
			return s;
		}
		return null;
	}

	public Switch getSwitch(Point switchPosition){
		Iterator<Switch> iterator = this.switches.iterator();
		while(iterator.hasNext()){
			Switch s = iterator.next();
			if(s.getPosition().x == switchPosition.x && s.getPosition().y == 
switchPosition.y){
				return s;
			}
		}
		return null;
	}

	public Led getLed(Point ledPosition){
		Iterator<Led> iterator = this.leds.iterator();
		while(iterator.hasNext()){
			Led l = iterator.next();
			if(l.getPosition().x == ledPosition.x && l.getPosition().y ==
ledPosition.y){
				return l;
			}
		}
		return null;
	}

	// setter & getter

	@Override
	public Point getPosition() {
		return position;
	}

	@Override
	public void setPosition(Point point) {
		this.position = point;
	}

	public void setWires(ArrayList<Wire> wires) {
		this.wires = wires;
	}

	public ArrayList<Wire> getWires() {
		return wires;
	}

	public ArrayList<Socket> getInputs() {
		return inputs;
	}

	public void setInputs(ArrayList<Socket> inputs) {
		this.inputs = inputs;
	}

	public void setSwitches(ArrayList<Switch> switches) {
		this.switches = switches;
	}

	public ArrayList<Switch> getSwitches() {
		return switches;
	}

	public ArrayList<Socket> getOutputs() {
		return outputs;
	}

	public void setOutputs(ArrayList<Socket> outputs) {
		this.outputs = outputs;
	}

	public void setLeds(ArrayList<Led> leds) {
		this.leds = leds;
	}
```

```
       public ArrayList<Led> getLeds() {
              return leds;
       }

       public Socket getGround() {
              return ground;
       }

       public void setGround(Socket ground) {
              this.ground = ground;
       }

       public Socket getPower() {
              return power;
       }

       public void setPower(Socket power) {
              this.power = power;
       }
}
------------------------------------------------------------------------
/*
A Library class used for opening a browser to view help document online
*/
------------------------------------------------------------------------

import java.lang.reflect.Method;
import javax.swing.JOptionPane;
import java.util.Arrays;

/**
 * <b>Bare Bones Browser Launch for Java</b><br>
 * Utility class to open a web page from a Swing application
 * in the user's default browser.<br>
 * Supports: Mac OS X, GNU/Linux, Unix, Windows XP/Vista<br>
 * Example Usage:<code><br> &nbsp; &nbsp;
 *    String url = "http://www.google.com/";<br> &nbsp; &nbsp;
 *    BareBonesBrowserLaunch.openURL(url);<br></code>
 * Latest Version: <a
href="http://www.centerkey.com/java/browser/">www.centerkey.com/java/browser</a><br>
 * Author: Dem Pilafian<br>
 * Public Domain Software -- Free to Use as You Like
 * @version 2.0, May 26, 2009
 */
public class BareBonesBrowserLaunch {

   static final String[] browsers = { "firefox", "opera", "konqueror", "epiphany",
      "seamonkey", "galeon", "kazehakase", "mozilla", "netscape" };

   /**
    * Opens the specified web page in a web browser
    * @param url A web address (URL) of a web page (ex: "http://www.google.com/")
    */
   public static void openURL(String url) {
      String osName = System.getProperty("os.name");
      try {
         if (osName.startsWith("Mac OS")) {
            Class<?> fileMgr = Class.forName("com.apple.eio.FileManager");
            Method openURL = fileMgr.getDeclaredMethod("openURL",
               new Class[] {String.class});
            openURL.invoke(null, new Object[] {url});
            }
         else if (osName.startsWith("Windows"))
            Runtime.getRuntime().exec("rundll32 url.dll,FileProtocolHandler " + url);
         else { //assume Unix or Linux
            boolean found = false;
            for (String browser : browsers)
               if (!found) {
                  found = Runtime.getRuntime().exec(
                     new String[] {"which", browser}).waitFor() == 0;
                  if (found)
                     Runtime.getRuntime().exec(new String[] {browser, url});
                  }
            if (!found)
               throw new Exception(Arrays.toString(browsers));
            }
         }
      catch (Exception e) {
         JOptionPane.showMessageDialog(null,
            "Error attempting to launch web browser\n" + e.toString());
         }
      }

   }

-------------------------------------------------------------------------
/*
ColorWithAName extends String class adding a color attribute to a normal String object.
It provides a method to get string attribute of the class.
*/
-------------------------------------------------------------------------
import java.awt.Color;


public class ColorWithAName extends Color{
       private static final long serialVersionUID = 341234;
       String name;
       public ColorWithAName(Color color, String name){
              super(color.getRGB());
              this.name = name;
       }
       public String toString(){
              return name;
       }
}
-------------------------------------------------------------------------
/*
CreditsPanel is a panel that holds the information related to developers and also their
pictures. The pictures together with the information are shown as a panel.
*/
-------------------------------------------------------------------------
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class CreditsPanel extends JPanel implements MouseListener{

       /**
        *
        */
       private static final long serialVersionUID = 1L;
       //Personal information constants
```

```
	private final String[ ] bbsimString = { "BBSim",	"BreadBoard Simulation 
Tool", "© 2009", "", "", "", ""};
	private final String[ ] mustafaString = { "Mustafa Zengin", 
"mzengin88@gmail.com",	"Bilkent University", "	Computer Engineering", 
"www.mustafazengin.info", "", ""};
	private final String[ ] korpeString = { "Erdin\u00E7 K\u00F6rpeo\u011Flu", 
"e.korpeoglu@gmail.com", "Bilkent University", "	Computer Engineering", "", "", ""};
	private final String[ ] battalString = { "Mustafa Battal", 
"mbattal1990@gmail.com", "Bilkent University", "	Computer Engineering", "", "", ""};
	private final String[ ] salimString = {"Salim Sar\u0131murat", 
"sarimura@gmail.com", "Bilkent University", "	Computer Engineering", 
"http://salim.sarimurat.net", "", ""};

	//You can determine the colors for each line of information of each person
	private final Color[ ] bbsimColors = {Color.GREEN, Color.BLUE, Color.BLACK, 
Color.BLUE, Color.BLACK, Color.BLUE, Color.BLACK };
	private final Color[ ] mustafaColors = {Color.BLACK, Color.BLACK, Color.BLACK, 
Color.BLACK, Color.BLACK, Color.BLACK, Color.BLACK };
	private final Color[ ] korpeColors = {Color.BLACK, Color.BLACK, Color.BLACK, 
Color.BLACK, Color.BLACK, Color.BLACK, Color.BLACK };
	private final Color[ ] battalColors = {Color.BLACK, Color.BLACK, Color.BLACK, 
Color.BLACK, Color.BLACK, Color.BLACK, Color.BLACK };
	private final Color[ ] salimColors = {Color.BLACK, Color.BLACK, Color.BLACK, 
Color.BLACK, Color.BLACK, Color.BLACK, Color.BLACK };

	//constants for the paths of coders' pictures
	private final String bbsimPicPath = "img/credits/bbsim.png";
	private final String mustafaPicPath = "img/credits/mustafa2.png";
	private final String korpePicPath = "img/credits/korpe2.png";
	private final String battalPicPath = "img/credits/battal2.png";
	private final String salimPicPath = "img/credits/salim2.png";

	JPanel titlePanel;
		JLabel title;

	JPanel picturePanel;
		JLabel mustafa;
		JLabel korpe;
		JLabel battal;
		JLabel salim;

	JPanel infoPanel;
		ImageIcon picture;
		JPanel infoPicPanel;
		JLabel pictureLabel;
		JPanel info2Panel;
		JLabel[ ] infoLine;

	JPanel backPanel;
		JButton	backButton;
		JPanel pHolder;

	Font font;


	public CreditsPanel(){
	prepareGPanels( );
    }

    public void preparePanels( ){

	//fonts for information
```

```
        font = ( new Font( Font.SANS_SERIF, Font.PLAIN, 18));

        FlowLayout layout;
                layout = new FlowLayout( );
                layout.setVgap( 0);
                layout.setHgap( 0);
                setLayout( layout);
                setBackground( Color.WHITE);

                //picturePanel preparation - mini pictures at the top
        picturePanel = new JPanel( );
        picturePanel.setPreferredSize( new Dimension( 800, 200));
        picturePanel.setLayout( new GridLayout( 1, 4));

        mustafa = new JLabel( new ImageIcon( "img/credits/mustafa.png"));
        korpe = new JLabel ( new ImageIcon( "img/credits/korpe.png"));
        battal = new JLabel( new ImageIcon( "img/credits/battal.png"));
        salim = new JLabel( new ImageIcon( "img/credits/salim.png"));

        mustafa.addMouseListener( this);
        korpe.addMouseListener( this);
        battal.addMouseListener( this);
        salim.addMouseListener( this);

                picturePanel.add( mustafa);
                picturePanel.add( korpe);
                picturePanel.add( battal);
                picturePanel.add( salim);

                //information Panel preperation
                infoPanel = new JPanel( );
                infoPanel.setBackground( Color.WHITE);

                //pics of the coders
                infoPicPanel = new JPanel( );
                infoPicPanel.setPreferredSize( new Dimension( 360, 320));
                infoPicPanel.setBackground( Color.WHITE);

                //text info about the coders
                infoLine = new JLabel[ 7];
                info2Panel = new JPanel( );
                info2Panel.setPreferredSize( new Dimension( 400, 320));
                info2Panel.setBackground( Color.WHITE);
                info2Panel.setLayout( new GridLayout( 8, 1));

                //add info label by label
                picture = new ImageIcon( );
                for ( int i = 0; i < infoLine.length; i++)
                {
                        infoLine[ i] = new JLabel( );
                }

                //there is two the same repeated loop, because the following line must be
between those.
                setInfoPanel( bbsimString, bbsimColors, bbsimPicPath);

                for ( int i = 0; i < infoLine.length; i++)
                {
                        infoLine[ i].setFont( font);
                        info2Panel.add( infoLine[ i]);
                }
```

```
              pictureLabel = new JLabel ( picture);
              pictureLabel.setPreferredSize( new Dimension ( 360, 320));
              infoPicPanel.add( pictureLabel);

              infoPanel.add( infoPicPanel);
              infoPanel.add( info2Panel);
              infoPanel.setPreferredSize( new Dimension( 800, 320));

              add( picturePanel);
        add( infoPanel);
     }
     /**
      *this method changes the info, it will be used in MouseListeners
      */
     public void setInfoPanel( String[ ] details, Color[ ] colors, String pictFilePath)
     {
        for( int i = 0; i < infoLine.length; i++)
        {
              infoLine[ i].setText( details[ i]);
              infoLine[ i].setForeground( colors[ i]);
              picture.setImage( ( new ImageIcon( pictFilePath).getImage( )));
              repaint( );
        }
     }

     public void mouseEntered( MouseEvent e){
        //determine the screen to hold info of the coder whose little pic is rollovered
by mouse.
        if( e.getSource( ) == mustafa){
              setInfoPanel( mustafaString, mustafaColors, mustafaPicPath);
        }
        else if( e.getSource( ) == korpe){
              setInfoPanel( korpeString, korpeColors, korpePicPath);
        }
        else if( e.getSource( ) == battal){
              setInfoPanel( battalString, battalColors, battalPicPath);
        }
        else if( e.getSource( ) == salim){
              setInfoPanel( salimString, salimColors, salimPicPath);
        }
        else{
              setInfoPanel( bbsimString, bbsimColors, bbsimPicPath);
        }
     }

     public void mouseExited( MouseEvent e){
        //when mouse is exited from the little pic panel, change screen to default info
        setInfoPanel( bbsimString, bbsimColors, bbsimPicPath);
     }

     //following methods are necessarily overridden because of implementation of the
listener
     public void mouseClicked( MouseEvent e){
     }

     public void mouseReleased( MouseEvent e){

     }

     public void mousePressed( MouseEvent e){

     }
```

```
}

--------------------------------------------------------------------------
/*
This class is a panel which generates user interface for multiple number of kmaps. It
generates functions using selected 1's from the KMaps by the user. It verifies the
generated functions with the values entered to the truth table by the user. It also has
many private operations dealing with the algorithm working behind the function
generation and verification processes. Among its attributes, there are the functions
generated from K-Maps, selected 1's from the Kmaps, and user interface elements such as
panels, labels buttons etc.
*/
--------------------------------------------------------------------------
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.ArrayList;
import java.util.Collections;
import java.util.StringTokenizer;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.border.Border;

public class EditKMapPanel extends JPanel implements MouseListener, ActionListener{
	/**
	 *
	 */
	private static final long serialVersionUID = 98034;
	int noOfInputs;
	int noOfOutputs;
	int rows;
	int columns;
	String[] inputNames;
	String[] outputNames;
	JPanel[] kmapPanels;
	JPanel[] kmapOuterPanels;
	JPanel[] buttonPanels;
	JPanel[] functionPanels;
	JButton doneButton;
	JButton verifyButton;
	JButton[] getFunctionButtons;
	JTextField[] functions;
	JTextField[] outputs;
	JLabel[][][] labels;
	JPanel[][][] panels;
	JPanel kms;
	JPanel verifyPanel;
	JScrollPane pane;
	JPanel panePanel;
	boolean[][][] selected;
	int[][] data;
	final int LABEL_SIZE = 50;
	final Color[] COLORS = { new Color(120,40,0,30), new Color(40,120,0,30), new
Color(0,40,120,50), new Color(160,0,0,30), new Color(0,160,0,30), new
Color(0,0,160,30)};
	int[] colorPointer;
	Border selectedBorder;
	Border unselectedBorder;
	public EditKMapPanel(String[] inputNames, String[] outputNames, int[][] data){
		this.inputNames = inputNames;
		this.outputNames = outputNames;
		this.noOfInputs = inputNames.length;
		this.noOfOutputs = outputNames.length;
		this.rows = noOfInputs == 3 ? 2 : noOfInputs;
		this.columns = noOfInputs % 2 == 1 ? noOfInputs + 1 :noOfInputs;
		this.labels = new JLabel[noOfOutputs][rows][columns];
		this.panels = new JPanel[noOfOutputs][rows][columns];
		this.selected = new boolean[noOfOutputs][rows][columns];
		this.kmapPanels = new JPanel[noOfOutputs];
		this.kmapOuterPanels = new JPanel[noOfOutputs];
		this.buttonPanels = new JPanel[noOfOutputs];
		this.functionPanels = new JPanel[noOfOutputs];
		this.functions = new JTextField[noOfOutputs];
		this.outputs = new JTextField[noOfOutputs];
		this.getFunctionButtons = new JButton[noOfOutputs];
		this.doneButton = new JButton("Done!");
		this.verifyButton = new JButton("verify");
		verifyButton.addActionListener(this);
		this.colorPointer = new int[noOfOutputs];
		this.unselectedBorder = BorderFactory.createRaisedBevelBorder();
		this.selectedBorder = BorderFactory.createLoweredBevelBorder();
		this.verifyPanel = new JPanel();
		this.kms = new JPanel();
		kms.setLayout(new GridLayout((noOfOutputs+1)/2, 2));
		for(int i = 0; i < noOfOutputs; i++){
			colorPointer[i] = 0;
			functionPanels[i] = new JPanel();
			functions[i] = new JTextField("");
			outputs[i] = new JTextField(outputNames[i] + "= ");
			outputs[i].setSize(500, 15);
			outputs[i].setEditable(false);
			functions[i].setEditable(false);
			functionPanels[i].setLayout(new BorderLayout());
			functionPanels[i].add(outputs[i],BorderLayout.WEST);
			functionPanels[i].add(functions[i],BorderLayout.CENTER);

			getFunctionButtons[i] = new JButton("Group!");
			getFunctionButtons[i].addActionListener(this);
			buttonPanels[i] = new JPanel();
			buttonPanels[i].add(getFunctionButtons[i]);

			kmapPanels[i] = new JPanel();
			kmapPanels[i].setLayout(new GridLayout(rows,columns));
			kmapPanels[i].setSize(columns * LABEL_SIZE, rows * LABEL_SIZE);
			kmapPanels[i].setPreferredSize(new Dimension(columns * LABEL_SIZE,
rows * LABEL_SIZE));

			kmapOuterPanels[i] = new JPanel();
			kmapOuterPanels[i].setBorder(BorderFactory.createTitledBorder("Kmap
for " + outputNames[i]));
			kmapOuterPanels[i].setLayout(new BorderLayout());
```

```
                    //kmapOuterPanels[i].setSize(columns * LABEL_SIZE, rows *
LABEL_SIZE);
                    //kmapOuterPanels[i].setPreferredSize(new Dimension(columns *
LABEL_SIZE, rows * LABEL_SIZE));

                    kmapOuterPanels[i].add(kmapPanels[i], BorderLayout.CENTER);
                    JPanel holder = new JPanel();
                    holder.setLayout(new GridLayout(2,1));
                    holder.add(functionPanels[i]);
                    holder.add(buttonPanels[i]);
                    holder.setBackground(Color.WHITE);
                    kmapOuterPanels[i].add(holder, BorderLayout.SOUTH);

             }
             for(int i = 0; i < noOfOutputs; i++){
                    for(int j = 0; j < rows; j++){
                           for(int k = 0; k < columns; k++){
                                  labels[i][j][k] = new
JLabel(Integer.toString(data[numbering(j,k)][i]));
                                  labels[i][j][k].addMouseListener(this);
                                  panels[i][j][k] = new JPanel();
                                  GridLayout panelL = new GridLayout(1,1);
                                  panelL.setHgap(0);
                                  panelL.setVgap(0);
                                  panels[i][j][k].setLayout(panelL);
                                  panels[i][j][k].add(labels[i][j][k]);

                                  panels[i][j][k].setPreferredSize(new
Dimension(LABEL_SIZE, LABEL_SIZE));
                                  labels[i][j][k].setPreferredSize(new
Dimension(LABEL_SIZE, LABEL_SIZE));
                                  labels[i][j][k].setBorder(unselectedBorder);
                                  labels[i][j][k].setForeground(Color.BLACK);
                                  labels[i][j][k].setBackground(Color.WHITE);
                                  selected[i][j][k] = false;
                                  kmapPanels[i].add(panels[i][j][k]);
                                  kmapPanels[i].setBackground(Color.WHITE);
                           }
                    }
                    kmapOuterPanels[i].setBackground(Color.WHITE);
                    kms.add(kmapOuterPanels[i]);

             }
             verifyPanel.add(verifyButton);
             panePanel = new JPanel();
             panePanel.setLayout(new GridLayout(2,1));
             panePanel.add(kms);
             panePanel.add(verifyPanel);
             pane = new JScrollPane(panePanel, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
             pane.setSize(650,400);
             pane.setPreferredSize(new Dimension(650,400));
             pane.getVerticalScrollBar().setValue(0);
             add(pane);
      }

      public boolean isSelectable(ArrayList<Integer> locations){
             Collections.sort(locations);
             boolean isSelectable = false;
             switch(locations.size()){
                    case(1): isSelectable = true; break;
                    case(2): isSelectable = size2Selectable(locations); break;
```

```
                case(4): isSelectable = size4Selectable(locations); break;
                case(8): isSelectable = size8Selectable(locations); break;
                case(16): isSelectable = true; break;
        }
        return isSelectable;
}

private boolean size2Selectable(ArrayList<Integer> locations){
        int[][] size2 = {   {0,1},
                                        {0,2},
                                        {0,4},
                                        {0,8},
                                        {1,3},
                                        {1,5},
                                        {1,9},
                                        {2,3},
                                        {2,6},
                                        {2,10},
                                        {3,7},
                                        {3,11},
                                        {4,5},
                                        {4,6},
                                        {4,12},
                                        {5,7},
                                        {5,13},
                                        {6,7},
                                        {6,14},
                                        {7,15},
                                        {8,9},
                                        {8,12},
                                        {8,10},
                                        {9,11},
                                        {9,13},
                                        {10,11},
                                        {10,14},
                                        {11,15},
                                        {12,13},
                                        {12,14},
                                        {13,15},
                                        {14,15},      };
        for(int i= 0; i< size2.length; i++){
                int count = 0;
                for(int j = 0; j < 2; j++){
                        if(size2[i][j] == locations.get(j))
                                count++;
                        else
                                count = 0;
                }
                if(count == 2)
                        return true;
        }
        return false;
}

private boolean size4Selectable(ArrayList<Integer> locations){
        int[][] size4 = {   {0,1,2,3},
                                        {0,1,8,9},
                                        {1,3,9,11},
                                        {2,3,10,11},
                                        {0,2,8,10},
                                        {0,1,8,9},
                                        {4,5,6,7},
                                            {12,13,14,15},
                                            {8,9,10,11},
                                            {0,4,8,12},
                                            {1,5,9,13},
                                            {3,7,11,15},
                                            {2,6,10,14},
                                            {0,1,4,5},
                                            {1,3,5,7},
                                            {2,3,6,7},
                                            {0,2,4,6},
                                            {4,5,12,13},
                                            {5,7,13,15},
                                            {6,7,14,15},
                                            {4,6,12,14},
                                            {8,9,12,13},
                                            {9,11,13,15},
                                            {10,11,14,15},
                                            {8,10,12,14}};
                for(int i= 0; i< size4.length; i++){
                        int count = 0;
                        for(int j = 0; j < 4; j++){
                                if(size4[i][j] == locations.get(j))
                                        count++;
                                else
                                        count = 0;
                        }
                        if(count == 4)
                                return true;
                }
                return false;
        }

        private boolean size8Selectable(ArrayList<Integer> locations){
                int[][] size8 = {   {0,1,2,3,4,5,6,7},
                                            {4,5,6,7,12,13,14,15},
                                            {8,9,10,11,12,13,14,15},
                                            {0,1,2,3,8,9,10,11},
                                            {0,2,4,6,8,10,12,14},
                                            {1,3,5,7,9,11,13,15},
                                            {2,3,6,7,10,11,14,15},
                                            {0,1,4,5,8,9,12,13}};
                for(int i= 0; i< size8.length; i++){
                        int count = 0;
                        for(int j = 0; j < 8; j++){
                                if(size8[i][j] == locations.get(j))
                                        count++;
                                else
                                        count = 0;
                        }
                        if(count == 8)
                                return true;
                }
                return false;
        }

        private int numbering(int i, int j){
                i = swapper(i);
                j = swapper(j);
                String binary = Integer.toBinaryString(i);
                binary += (columns > 2 && j < 2) ? "0" + Integer.toBinaryString(j) :
Integer.toBinaryString(j);
                return Integer.valueOf(binary, 2);
```

```
	}

	private String getFunction(ArrayList<Integer> selectedItems){
		ArrayList<String> strings = new ArrayList<String>();
		for(int i = 0; i < selectedItems.size(); i++){

	strings.add(stringExtender(Integer.toBinaryString(selectedItems.get(i))));
		}
		return functionGenerator(strings);
	}

	private String stringExtender(String input){
		String result = input;
		for(int i = input.length(); i < noOfInputs; i++){
			result = "0" + result;
		}
		return result;
	}

	private String functionGenerator(ArrayList<String> inputs){
		ArrayList<Integer> locations = new ArrayList<Integer>();
		char a;
		for(int i = 0; i < inputs.get(0).length(); i++){
			a = inputs.get(0).charAt(i);
			for(int j = 1; j < inputs.size(); j++){
				if(inputs.get(j).charAt(i) != a)
					if(locations.indexOf(i) < 0)
						locations.add(i);
			}
		}
		String functionalValue = "";
		for(int i = 0; i < noOfInputs; i++){
			if(!locations.contains(i))
				functionalValue += inputs.get(0).charAt(i) == '0' ?
inputNames[i] + "\'" : inputNames[i];
		}
		return functionalValue;
	}

	private int[] getLoc(int value){
		int[] result = new int[2];
		String binary = stringExtender(Integer.toBinaryString(value));
		switch(noOfInputs){
			case(1):{
				result[0] = 0;
				result[1] = Integer.parseInt(binary);
			}break;
			case(2):{
				result[0] = Integer.parseInt(binary.substring(0, 1), 2);
				result[1] = Integer.parseInt(binary.substring(1, 2), 2);
			}break;
			case(3):{
				result[0] = Integer.parseInt(binary.substring(0, 1), 2);
				result[1] = Integer.parseInt(binary.substring(1, 3), 2);
			}break;
			case(4):{
				result[0] = Integer.parseInt(binary.substring(0, 2), 2);
				result[1] = Integer.parseInt(binary.substring(2, 4), 2);
			}break;
		}
		result[0] = swapper(result[0]);
		result[1] = swapper(result[1]);
		return result;
	}

	private int swapper(int a){
		if(a == 3)
			a = 2;
		else if(a == 2)
			a = 3;
		return a;
	}

	private void changeSelection(int i, int j, int k){
		if(labels[i][j][k].getBorder() == unselectedBorder){
			labels[i][j][k].setBorder(selectedBorder);
		}
		else{
			labels[i][j][k].setBorder(unselectedBorder);
		}
		labels[i][j][k].setForeground(labels[i][j][k].getBackground().getRed() > 
240 || labels[i][j][k].getBackground().getGreen() > 240 ? Color.BLACK: Color.WHITE);
	}

	private static Color colorAdder(Color x, Color y){
		return new Color((x.getRed() + y.getRed()) % 256,(x.getBlue() + 
y.getBlue()) % 256, (x.getGreen() + y.getGreen()) % 256, x.getAlpha());
	}

	private static Color colorSubtructor(Color x, Color y){
		return new Color((x.getRed() - y.getRed() + 256) % 256,(x.getBlue() - 
y.getBlue() + 256) % 256, (x.getGreen() - y.getGreen() + 256) % 256, x.getAlpha());
	}

	@Override
	public void mouseClicked(MouseEvent e) {
		for(int i = 0; i < noOfOutputs; i++){
			for(int j = 0; j < rows; j++){
				for(int k = 0; k < columns; k++){
					if(labels[i][j][k] == e.getSource()){
						changeSelection(i, j, k);
					}
				}
			}
		}

	}

	private void removeAllSelections(int i){
		for(int j = 0; j < rows; j++)
			for(int k = 0; k < columns; k++){
				labels[i][j][k].setBorder(unselectedBorder);
			}
	}

	@Override
	public void mouseEntered(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseExited(MouseEvent arg0) {
		// TODO Auto-generated method stub
	}

	@Override
	public void mousePressed(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void mouseReleased(MouseEvent arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void actionPerformed(ActionEvent e) {
		if(e.getActionCommand().equals("verify")){
			String[] funcs = new String[noOfOutputs];
			for(int i = 0; i < noOfOutputs; i++)
				funcs[i] = functions[i].getText();
			System.out.println(verifyWithTruthTable(funcs, data, inputNames));
		}else{
			for(int i = 0; i < noOfOutputs; i++){
				if(e.getSource() == getFunctionButtons[i])
				{
					System.out.println(i + " th button clicked");
					int[][] dataX = {   {0,0,0,0,1},
										{0,0,1,0,1},
										{0,1,0,1,0},
										{0,1,1,0,1},
										{1,0,0,1,0},
										{1,0,1,1,0},
										{1,1,0,0,0},
										{1,1,1,1,1}};
					String[] func = {"A\'BC' + AB\' + AC", "A\'B\' +
BC"};
					String[] inputX = {"A","B", "C"};
					System.out.println(verifyWithTruthTable(func, dataX,
inputX));
					ArrayList<Integer> selectedItems = new
ArrayList<Integer>();
					for(int j = 0; j < rows; j++)
						for(int k = 0; k < columns; k++)
							if(labels[i][j][k].getBorder() ==
selectedBorder)

	selectedItems.add(numbering(j,k));
					if(isSelectable(selectedItems)){

						removeAllSelections(i);

	functions[i].setText(functions[i].getText().isEmpty() ?
getFunction(selectedItems) : functions[i].getText() + " + " +
getFunction(selectedItems));
						functions[i].repaint();
						functions[i].validate();
						for(int z = 0; z < selectedItems.size(); z++)

	panels[i][getLoc(selectedItems.get(z))[0]][getLoc(selectedItems.get(z))[1]].setB
ackground(colorAdder(panels[i][getLoc(selectedItems.get(z))[0]][getLoc(selectedItems.ge
t(z))[1]].getBackground(), COLORS[colorPointer[i]]));
```

```
                                        if(colorPointer[i] == COLORS.length - 1)
                                              colorPointer[i] = 0;
                                        else
                                              colorPointer[i]++;
                                  }
                                  else{
                                        JOptionPane optionPane = new JOptionPane();
                                        optionPane.createDialog(this,"Oops");/*

                                        JDialog message = new JDialog();
                                        message.setSize(300,80);
                                        message.setResizable(false);
                                        message.setTitle("deneme");
                                        message.add(new JLabel("The selection cannot be
grouped!"));
                                        message.setVisible(true);*/

                                  }
                                  return;
                            }
                      }
                }
          }

          private boolean verifyWithTruthTable(String[] functionStrings, int [][] allData,
String[] inputNames){
                StringTokenizer st;
                int totalLines =(int)Math.pow(2, inputNames.length);
                int count = 0;
                ArrayList<String> ands = new ArrayList<String>();
                ArrayList<String> andsBinary = new ArrayList<String>();

                for(int output = inputNames.length; output < inputNames.length + 2;
output++){
                      ands = new ArrayList<String>();
                      for(int i = 0; i < 2; i++){
                      st = new StringTokenizer(functionStrings[i], " + ");
                            while(st.hasMoreTokens()){
                                  ands.add(st.nextToken());
                                  andsBinary.add("");
                            }
                      }
                      for(int line = 0; line < totalLines; line++){
                            andsBinary = new ArrayList<String>();
                            for(int i = 0; i < ands.size(); i++)
                                  andsBinary.add("");
                            for(int i = 0; i < ands.size(); i++){
                                  for(int j = 0; j < inputNames.length; j++){
                                        andsBinary.set(i,
ands.get(i).replaceAll(inputNames[j] + "\'", allData[line][j] == 1 ? "0" : "1"));
                                        andsBinary.set(i,
ands.get(i).replaceAll(inputNames[j] , Integer.toString(allData[line][j])));
                                        andsBinary.set(i, and(andsBinary.get(i)));
                                  }

                            }
                      }
                }
                return count == 0 ? true : false;
          }
          private String and(String input){
                if(input.indexOf("0") > -1)
```

```
                    return "0";
            else
                    return "1";
        }
        private int or(ArrayList<String> input){
            if(input.contains("1"))
                    return 1;
            else
                    return 0;
        }
}
--------------------------------------------------------------------------
/*
This class provides a user interface to make the user see the default inputs and make
him set the output values accordingly. It has two operations to set output values to
all 1 or all 0. It also keeps track of data visualized as table and ready to be written
to the project as an attribute. The values determined in this panel forms the basis for
the verification in the following steps.
*/
--------------------------------------------------------------------------
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import javax.swing.table.*;


public class EditTTPanel extends JPanel implements ActionListener{
        /**
         *
         */
        private static final long serialVersionUID = 1L;
        JTable table;
        TableModel dataModel;
        DefaultTableCellRenderer cellRenderer;
        Object[][] data;
        int inputs, outputs, totalValues, totalIO;
        String[] inputNames, outputNames;
        TableColumn[] tableColumnsForOutputs;
        JComboBox combobox[];
        JButton doneButton;
        TableCellEditor cellEditor[];
        ColorWithAName output0, output1;
        ProjectControl projectControl;

        public EditTTPanel(final String[] inputNames, final String[] outputNames,
ProjectControl projectControl){
            this.projectControl = projectControl;
            this.inputs = inputNames.length;
            this.outputs = outputNames.length;
            this.totalIO = inputs + outputs;
            this.inputNames = inputNames;
            this.outputNames = outputNames;
            this.totalValues =(int)Math.pow(2, inputs);
            data = new Object[totalValues][totalIO];
            output1 = new ColorWithAName(Color.GREEN, "1");
            output0 = new ColorWithAName(Color.RED, "0");
            for(int i = 0; i < totalValues; i++){
                    for(int j = 0; j < inputs; j++){
                        data[i][inputs - j -1] = (i)%(int)Math.pow(2, j+1) >=
(int)Math.pow(2,j) ? new ColorWithAName(Color.YELLOW, "1") : new
ColorWithAName(Color.BLUE, "0");
                }
            }
            for(int i = 0; i < totalValues; i++){
                for(int j = inputs; j < totalIO; j++){
                    data[i][j] = output1;
                }
            }
            cellRenderer = new DefaultTableCellRenderer(){
                /**
                 *
                 */
                private static final long serialVersionUID = 1L;

                public void setValue(Object o){
                    if(o instanceof ColorWithAName) {
                        ColorWithAName a = (ColorWithAName)o;
                        setBackground(a);
                        setForeground((a.getRed() > 240 || a.getGreen() >
240)? Color.BLACK : Color.WHITE);
                        setText(a.toString());
                    }

                }
            };

            dataModel = new AbstractTableModel() {
                /**
                 *
                 // TODO Auto-generated method stub
            */
                private static final long serialVersionUID = 1L;
                public int getColumnCount() { return totalIO;}
                public int getRowCount() { return data.length;}
                public Object getValueAt(int row, int col) {return data[row][col];}
                public String getColumnName(int column) {return (column < inputs) ?
inputNames[column] : outputNames[column-inputs];}
                public boolean isCellEditable(int row, int col) {return col!=1;}
                public void setValueAt(Object aValue, int row, int column) {
                    data[row][column] = aValue;
                }
            };
            table = new JTable(dataModel);

            TableColumn[] tableColumns = new TableColumn[totalIO];
            for(int i = 0; i < inputs; i++){
                tableColumns[i] = table.getColumn(inputNames[i]);
                tableColumns[i].setCellRenderer(cellRenderer);
            }
            tableColumnsForOutputs = new TableColumn[outputs];
            combobox = new JComboBox[outputs];
            cellEditor = new TableCellEditor[outputs];
            for(int i = 0; i< outputs; i++){
                combobox[i] = new JComboBox();
                combobox[i].addItem(output1);
                combobox[i].addItem(output0);
                cellEditor[i] = new DefaultCellEditor(combobox[i]);
                tableColumnsForOutputs[i] = table.getColumn(outputNames[i]);
                tableColumnsForOutputs[i].setCellEditor(cellEditor[i]);
                tableColumnsForOutputs[i].setCellRenderer(cellRenderer);
            }
            setAllToOne();
            JScrollPane scrollPane = new JScrollPane(table);
            setLayout(new BorderLayout());
            add(scrollPane, BorderLayout.CENTER);
            JPanel buttonPanel = new JPanel();
            buttonPanel.setLayout(new GridLayout(1,3));
            JButton setToOneButton = new JButton("Set All to 1");
            JButton setToZeroButton = new JButton("Set All to 0");
            doneButton = new JButton("Done");
            doneButton.addActionListener(projectControl);
            setToOneButton.addActionListener(this);
            setToZeroButton.addActionListener(this);
            buttonPanel.add(setToOneButton);
            buttonPanel.add(setToZeroButton);
            buttonPanel.add(doneButton);
            add(buttonPanel, BorderLayout.SOUTH);
        }
        private void setAllToZero(){
            for(int i = 0; i< totalValues; i++)
                for(int j = inputs; j < totalIO; j++){
                    table.getModel().setValueAt(output0, i, j);
                }
            table.repaint();
            table.validate();
        }
        private void setAllToOne(){
            for(int i = 0; i< totalValues; i++)
                for(int j = inputs; j < totalIO; j++){
                    table.getModel().setValueAt(output1, i, j);
                }
            table.repaint();
            table.validate();
        }
        @Override
        public void actionPerformed(ActionEvent e) {
            if(e.getActionCommand().equals("Set All to 0"))
                setAllToZero();
            else if(e.getActionCommand().equals("Set All to 1"))
                setAllToOne();
        }
        public JButton getDoneButton(){
            return this.doneButton;
        }
}


------------------------------------------------------------------------
/*
This class is the user interface used as one of the project creation steps. The user
can determine the number of inputs and outputs together with the output names. It also
informs the ProjectControl to create project according to the infromation determined
using this panel.
*/
------------------------------------------------------------------------
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import javax.swing.table.*;
```

```
public class IODetermineForm extends JPanel implements ActionListener{
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	final int COL = 2;
	final int MAX_INPUTS = 4;
	final int MAX_OUTPUTS = 8;
	final String[] HEADER = {"IO Name","IO Type"};
	final String[] INPUTS = {"A", "B", "C", "D"};

	JPanel firstPanel;
	JComboBox noOfInputsCombo;
	JComboBox noOfOutputsCombo;
	JLabel noOfInputsLabel;
	JLabel noOfOutputsLabel;
	JButton nextButton;

	JPanel secondPanel;
	JComboBox initialCombo;
	JLabel initialStateLabel;
	JPanel secondOfSecondPanel;
	JTable table;
	TableModel dataModel;
	DefaultTableCellRenderer cellRenderer;
	JButton previousButton;
	JButton doneButton;
	ProjectControl projectControl;
	JFrame frame;

	int inputs, outputs, totalIO;
	Object data[][];
	public IODetermineForm(ProjectControl projectControl, JFrame frame){
		this.frame = frame;
		this.projectControl = projectControl;
		generateNoOfIO();
		add(firstPanel);
	}
	@Override
	public void actionPerformed(ActionEvent ae) {
		if(ae.getActionCommand().equals("Next>>")){
			this.inputs = noOfInputsCombo.getSelectedIndex() + 1;
			this.outputs = noOfOutputsCombo.getSelectedIndex() + 1;
			this.remove(firstPanel);
			generateIOTable();
			this.add(secondPanel);
		}
		else if(ae.getActionCommand().equals("<<Previous")){
			this.remove(secondPanel);
			validate();
			generateNoOfIO();
			noOfInputsCombo.setSelectedIndex(inputs-1);
			noOfOutputsCombo.setSelectedIndex(outputs-1);
			this.add(firstPanel);
		}
		else if(ae.getActionCommand().equals("Done")){
			boolean thereIsEmpty = false;
			boolean thereIsRepetition = false;
			String[] inputNames = new String[inputs];
			String[] outputNames = new String[outputs];
			for(int i = 0; i < this.inputs; i++)
```

```
                        inputNames[i]= data[i][0].toString();
                    for(int i = this.inputs; i < this.inputs + this.outputs; i++){
                        if(i + 1 != this.inputs + this.outputs)
                            for(int j = i + 1; j < this.inputs + this.outputs;
j++)
                                if(data[i][0].toString().equals(data[j][0]))
                                    thereIsRepetition = true;
                        if(data[i][0].toString().isEmpty())
                            thereIsEmpty = true;
                        outputNames[i-this.inputs]= data[i][0].toString();
                    }
                    if(thereIsEmpty){
                        JOptionPane.showMessageDialog(this, "You have to enter
name(s) for output(s)");
                    }
                    else if(thereIsRepetition){
                        JOptionPane.showMessageDialog(this, "There is a repetition
in the names of outputs");
                    }
                    else{
                        projectControl.getProject().setNoOfInputs(inputs);
                        projectControl.getProject().setNoOfOutputs(outputs);

        projectControl.getProject().setInitialState(initialCombo.getSelectedIndex());
                        projectControl.getProject().setInputNames(inputNames);
                        projectControl.getProject().setOutputNames(outputNames);
                        projectControl.getMainFrame().initializeComponents();
                        projectControl.getMainFrame().repaint();
                        projectControl.getMainFrame().validate();
                        frame.dispose();
                    }
                }
                repaint();
                validate();
        }
        private void generateIOTable(){
                totalIO = inputs + outputs;
                data = new Object[totalIO][COL];
                for(int i = 0; i < totalIO; i++){
                    data[i][0] = i < inputs ? INPUTS[i] : new String("");
                    data[i][1] = i < inputs ? new ColorWithAName(Color.BLUE, "Input") :
new ColorWithAName(Color.GREEN, "Output");
                }
                cellRenderer = new DefaultTableCellRenderer(){
                    /**
                     *
                     */
                    private static final long serialVersionUID = 1L;

                    public void setValue(Object o){
                        if(o instanceof ColorWithAName) {
                            ColorWithAName a = (ColorWithAName)o;
                            setBackground(a);
                            setForeground((a.getRed() > 240 || a.getGreen() >
240)? Color.BLACK : Color.WHITE);
                            setText(a.toString());
                        }
                    }
                };

                dataModel = new AbstractTableModel() {
                    /**
```

```
                         *
                         */
                        private static final long serialVersionUID = 1L;
                        public int getColumnCount() { return HEADER.length; }
                        public int getRowCount() { return data.length;}
                        public Object getValueAt(int row, int col) {return data[row][col];}
                        public String getColumnName(int column) {return HEADER[column];}
                        //public Class getColumnClass(int c) {return getValueAt(0,
c).getClass();}
                        public boolean isCellEditable(int row, int col) {return col!=1 &&
row >= inputs;}
                        public void setValueAt(Object aValue, int row, int column) {
                              data[row][column] = aValue;
                        }

                  };
                  table = new JTable(dataModel);
                  TableColumn column = table.getColumn(HEADER[1]);
                  column.setCellRenderer(cellRenderer);
                  JScrollPane scrollPane = new JScrollPane(table);

                  secondPanel = new JPanel();
                  secondPanel.setLayout(new GridLayout(2,1));

                  secondOfSecondPanel = new JPanel();
                  secondOfSecondPanel.setLayout(new FlowLayout());
                  previousButton = new JButton("<<Previous");
                  previousButton.addActionListener(this);
                  doneButton = new JButton("Done");
                  doneButton.addActionListener(this);
                  secondOfSecondPanel.add(previousButton);
                  secondOfSecondPanel.add(doneButton);

                  secondPanel.add(scrollPane);
                  secondPanel.add(secondOfSecondPanel);

            }
            private void generateNoOfIO(){
                  firstPanel = new JPanel();
                  firstPanel.setLayout(new GridLayout(4,2));
                  noOfInputsLabel = new JLabel("No of Inputs: ");
                  noOfOutputsLabel = new JLabel("No of Outputs: ");
                  initialStateLabel = new JLabel("Initial State: ");
                  initialCombo = new JComboBox();
                  noOfInputsCombo = new JComboBox();
                  noOfOutputsCombo = new JComboBox();
                  nextButton = new JButton("Next>>");
                  nextButton.addActionListener(this);

                  for(int i = 1; i <= MAX_INPUTS; i++){
                        noOfInputsCombo.addItem(i);
                  }
                  for(int i = 1; i <= MAX_OUTPUTS; i++){
                        noOfOutputsCombo.addItem(i);
                  }
                  initialCombo.addItem("Truth Table");
                  initialCombo.addItem("Circuit Schematic");
                  initialCombo.addItem("BreadBoard");

                  firstPanel.add(noOfInputsLabel);
                  firstPanel.add(noOfInputsCombo);
                  firstPanel.add(noOfOutputsLabel);
                  firstPanel.add(noOfOutputsCombo);
                  firstPanel.add(initialStateLabel);
                  firstPanel.add(initialCombo);
                  JLabel empty = new JLabel();
                  firstPanel.add(empty);
                  firstPanel.add(nextButton);
            }
      }
      ------------------------------------------------------------------------
      /*
      Main is the runnable class of overall project. It creates ProjectContol and MainFrame
      objects to run a new program. It also delegates ProjectControl to MainFrame to provide
      a bidirectional association.
      */
      ------------------------------------------------------------------------
      import javax.swing.JFrame;
      import javax.swing.SwingUtilities;
      import javax.swing.UIManager;

      import org.jvnet.substance.skin.SubstanceBusinessBlueSteelLookAndFeel;
      public class Main {

            /**
             * @param args
             */
            public static void main(String[] args) {
                  JFrame.setDefaultLookAndFeelDecorated(true);
                  SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                              try {
                                      UIManager.setLookAndFeel(new
      SubstanceBusinessBlueSteelLookAndFeel());
                              } catch (Exception e) {
                                      System.out.println("UI failed to initialize");
                              }
                              try{
                                      ProjectControl projectControl = new ProjectControl();
                                      MainFrame mainFrame = new MainFrame(projectControl);
                                      projectControl.setMainFrame(mainFrame);
                              }catch(Exception e){
                                      e.printStackTrace();
                              }
                        }
                  });
                /**
                  try{
                        JFrame frame = new JFrame();

            UIManager.setLookAndFeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel");
                        IODetermineForm io = new IODetermineForm();
                        frame.add(io);
                        frame.setSize(new Dimension(800,600));
                        frame.setVisible(true);


                  }
                  catch(Exception e){
                        e.printStackTrace();
                  }
                  try{

            UIManager.setLookAndFeel("com.sun.java.swing.plaf.gtk.GTKLookAndFeel");
```

```
					MainFrame frame = new MainFrame();

					frame.setVisible(true);
				}
				catch(Exception e){
					e.printStackTrace();
				}
				*/
		}

}


--------------------------------------------------------------------------
/*
Project class is the data class that keeps the data required to save a project to
enable user to load the project using the file saved before. It keeps the data in
private attributes and it also provides getter and setter operations to manage all
these attributes.
*/
--------------------------------------------------------------------------
import java.io.Serializable;
import java.util.ArrayList;
public class Project implements Serializable{
	/**
	 *
	 */
	private static final long serialVersionUID = 1234817;
	private int noOfInputs, noOfOutputs;
	private TruthTable truthTable;
	private String[] functions;
	private String[] outputNames;
	private String[] inputNames;
	private CircuitSchematic cs;
	private BreadBoard bb;
	private ArrayList<Gate> gates;
	private int initialState;
	private int currentState;
	private String url;

	public Project(){}
	public Project(int noOfInputs, int noOfOutputs, String[] inputNames, String[]
outputNames, String url){
		this.noOfInputs = noOfInputs;
		this.noOfOutputs = noOfOutputs;
		this.inputNames = inputNames;
		this.outputNames = outputNames;
		this.url = url;
	}
	public int getInitialState() {
		return initialState;
	}
	public int getCurrentState() {
		return currentState;
	}
	public int getNoOfInputs() {
		return noOfInputs;
	}
	public int getNoOfOutputs() {
		return noOfOutputs;
	}
	public TruthTable getTruthTable() {
		return truthTable;
	}
	public String[] getFunctions() {
		return functions;
	}
	public String[] getOutputNames() {
		return outputNames;
	}
	public String[] getInputNames() {
		return inputNames;
	}
	public CircuitSchematic getCs() {
		return cs;
	}
	public BreadBoard getBb() {
		return bb;
	}
	public ArrayList<Gate> getGates() {
		return gates;
	}
	public String getUrl() {
		return url;
	}
	public void setUrl(String url) {
		this.url = url;
	}
	public void setNoOfInputs(int noOfInputs) {
		this.noOfInputs = noOfInputs;
	}
	public void setNoOfOutputs(int noOfOutputs) {
		this.noOfOutputs = noOfOutputs;
	}
	public void setTruthTable(TruthTable truthTable) {
		this.truthTable = truthTable;
	}
	public void setFunctions(String[] functions) {
		this.functions = functions;
	}
	public void setOutputNames(String[] outputNames) {
		this.outputNames = outputNames;
	}
	public void setInputNames(String[] inputNames) {
		this.inputNames = inputNames;
	}
	public void setCs(CircuitSchematic cs) {
		this.cs = cs;
	}
	public void setBb(BreadBoard bb) {
		this.bb = bb;
	}
	public void setGates(ArrayList<Gate> gates) {
		this.gates = gates;
	}
	public void setInitialState(int initialState) {
		this.initialState = initialState;
	}
	public void setCurrentState(int currentState) {
		this.currentState = currentState;
	}

}
-------------------------------------------------------------------------
```

```
/*
This class is the main control object that controls the overall project using both data
and user interface classes. It is also the facade class of the applied facade pattern.
As attributes, it has MainFrame object to manage the user interface classes and Project
object to manage data classes. It also has an access to XMLStorage class for save and
load operations.
*/
--------------------------------------------------------------------------
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ProjectControl implements ActionListener{
       private Project project;
       private MainFrame mainFrame;
       public ProjectControl(){
              this.project = new Project();
       }
       public Project getProject() {
              return project;
       }
       public MainFrame getMainFrame() {
              return mainFrame;
       }
       public void setMainFrame(MainFrame mainFrame) {
              this.mainFrame = mainFrame;
       }
       public void setProject(Project project) {
              this.project = project;
       }
       public void saveProject(){
              XMLStorage.saveProject(project, project.getUrl());
       }
       public Project loadProject(String url){
              return XMLStorage.loadProject(url);
       }
       @Override
       public void actionPerformed(ActionEvent e) {
              if(e.getSource() ==
((EditTTPanel)this.mainFrame.truthTablePanel).doneButton){
                     int[][] data = new int[(int)Math.pow(2,
project.getNoOfInputs())][project.getNoOfOutputs()];
                     for(int i = 0; i < data.length ; i++)
                            for(int j = 0; j < data[0].length; j++)
                                   data[i][j] =
Integer.parseInt(((ColorWithAName)((EditTTPanel)this.mainFrame.truthTablePanel).data[i]
[j+ project.getNoOfInputs()]).toString());

                     mainFrame.setKMapPanel(new EditKMapPanel(project.getInputNames(),
project.getOutputNames(), data));
                     mainFrame.tabs.setSelectedIndex(1);
                     mainFrame.repaint();
                     mainFrame.validate();
              }
       }
       private int[][] transpose(int[][] data){
              int[][] result = new int[data.length][data[0].length];
              for(int i = 0; i < data.length; i++)
                     for(int j = 0; j < data[0].length; j++)
                            result[i][j] = data[j][i];
              return result;
       }
}
-------------------------------------------------------------------------
/*
This class has two static methods, one of which is loading an existing project from a
file and the other is saving the current open project to a file.
*/
-------------------------------------------------------------------------
import java.beans.XMLDecoder;
import java.beans.XMLEncoder;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class XMLStorage {
	public static boolean saveProject(Project project, String url){
		try{
			XMLEncoder e = new XMLEncoder(
			   new BufferedOutputStream(
			       new FileOutputStream(url)));
			e.writeObject(project);
			e.close();
			return true;
		}catch(Exception exp){
			exp.printStackTrace();
			return false;
		}
	}
	public static Project loadProject(String url){
		Project result = new Project();
		try{
			XMLDecoder d = new XMLDecoder(
					new BufferedInputStream(
							new FileInputStream(url)));
			result = (Project)d.readObject();
			d.close();
		}catch(Exception exp){
			exp.printStackTrace();
		}
		return result;
	}
}


-------------------------------------------------------------------------
/*
Class for KMap entity object. It holds values in a boolean array
*/
-------------------------------------------------------------------------

public class KMap {
	private boolean[][] values;
	public KMap(int row, int col){
		this.values = new boolean[row][col];
	}
	public int getValueAt(int row, int col){
		return values[row][col] ? 1 : 0;
	}
	public void setValueAt(int row, int col, int value){
		values[row][col] = (value == 1) ? true : false;
	}
}
```

```
/* -----------------------------------------------------------------------
Object for fetching url of help file.
*/
-------------------------------------------------------------------------

public class HelpDataFetcher {
	private final static String ONLINE_URL = "";
	private final static String OFFLINE_URL = "/help/index.html";

	public static boolean isConnectionAvailable(){
		return false;
	}
	public static String getOnlineHelp(){
		return ONLINE_URL;
	}
	public static String getOfflineHelp(){
		return OFFLINE_URL;
	}

}
/*------------------------------------------------------------------------

Panel to show user help information
-----------------------------------------------------------------------*/

import java.awt.*;
import java.io.*;
import java.net.*;

import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.html.*;

public class HelpDisplayPanel extends JPanel {
	private static final long serialVersionUID = 857435;
	JEditorPane html;
	HelpDataFetcher dataFetcher;
	public HelpDisplayPanel(){
		setLayout(new BorderLayout());
		try {
			URL url = null;
			String path = null;
			try {
				if(dataFetcher.isConnectionAvailable())
					path = dataFetcher.getOnlineHelp();
				else
					path =  dataFetcher.getOfflineHelp();
				url = getClass().getResource(path);
			} catch (Exception e) {
				System.err.println("Failed to open " + path);
				url = null;
			}
			if(url != null) {
				html = new JEditorPane(url);
				html.setEditable(false);
				html.addHyperlinkListener(createHyperLinkListener());

				JScrollPane scroller = new JScrollPane();
				JViewport vp = scroller.getViewport();
				vp.add(html);
				add(scroller, BorderLayout.CENTER);
			}
			} catch (MalformedURLException e) {
				System.out.println("Malformed URL: " + e);
			} catch (IOException e) {
				System.out.println("IOException: " + e);
			}
		}
		public HyperlinkListener createHyperLinkListener() {
			return new HyperlinkListener() {
				public void hyperlinkUpdate(HyperlinkEvent e) {
					if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED)
{
						if (e instanceof HTMLFrameHyperlinkEvent)
{((HTMLDocument)html.getDocument()).processHTMLFrameHyperlinkEvent((HTMLFrameHyperlinkE
vent)e);
						} else {
							try {
								html.setPage(e.getURL());
							} catch (IOException ioe) {
								System.out.println("IOE: " + ioe);
							}
						}
					}
				}
			};
		}
}
-------------------------------------------------------------------------
```