Design Document Report
Drawing Editor

Comprehensive Design Document for a Drawing Editor System

DASS Bonus Assignment

Atidipt Ashnin [2022111020]
Harpreet Singh [2022101048]
Prakhar Jain [2022115006]
Prakhar Singhal [2022111025]
Ravessh Vyas [2022114002]

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Submitted May 6, 2024

Table of contents

Chapter 1

Introduction

1.1 Overview of the Drawing Editor

The drawing program aims to provide a user-friendly interface for creating and manipulating various types of drawing objects, such as lines, rectangles, polylines, curves, text, and images. It supports basic operations like drawing, selecting, deleting, copying, moving, and editing objects. Additionally, it allows grouping objects into larger units and provides options for saving and restoring drawings in a file format. The program also allows exporting drawings to XML for use with other applications. Its design is extensible, allowing for the integration of new object types, operations, editors, and file formats in the future.

1.2 Purpose of the Document

The purpose of the drawing program is to provide a versatile tool for creating and manipulating drawings with ease. It caters to users who need to create simple to moderately complex diagrams, sketches, or illustrations. By offering a range of object types and editing capabilities, the program enables users to express their ideas visually. The program's extensibility ensures that it can evolve to meet changing requirements and accommodate additional features or integration with other systems.

1.3 Scope of the System

The scope of the drawing program includes supporting an indefinite number of drawing object types, including lines and rectangles initially, with potential for expansion to include other types such as curves, text, and images. Users can draw objects using a mouse and manipulate them by selecting, deleting, copying, moving, or editing. The program allows grouping objects into larger units, exporting drawings to XML, and saving/restoring drawings from files. It supports basic operations like color selection, corner styles, and file management. The program aims to ensure usability within a canvas of at least 400 by 400 pixels, with optional support for larger canvases or scrolling.

Chapter 2

System Overview

2.1 Description of the Drawing Editor System

The Drawing Editor System is a software designed to ease the load of making Designs in System Design. It gives you the freedom to choose between different shapes and the ability to edit them. It allows you to make a group of different shapes for future use. Overall, it streamlines shape formation and enhances efficiency.

2.2 High-Level Architecture

The high-level architecture of the Drawing Editor System consists of interconnected modules designed to ensure seamless information flow and operation throughout the institution. Each module serves specific functions and interacts with others to facilitate efficient management of various tasks.

Chapter 3

Functional Requirements

3.1 Drawing Object Types

The program should support drawing various object types, including lines, rectangles, polylines, curves, text, and images. Users should be able to select an object type from a menu or toolbar and draw it using a mouse.

3.2 Manipulating Objects

Users should be able to manipulate objects by selecting them with the mouse, providing visual indication of selection. Supported operations include deletion, copying, moving, and editing of selected objects.

3.3 Editing Objects

Objects should be editable through dialog boxes that allow users to define properties such as color, line style, and corner style. Editing operations should be intuitive and provide feedback to users about changes made.

3.4 Grouping Objects

Objects can be grouped into larger units, allowing for collective manipulation. Grouped objects should maintain relative positions and orientations when moved or copied. Groups can be selected, ungrouped, or edited as needed.

3.5 Saving and Restoring Drawings

The program should support saving drawings to a file in ASCII text format and restoring them from files. Users can open and save drawings using menu options or toolbar buttons.

Automatic warnings should prevent users from losing unsaved work.

3.6 Export to XML

Users should have the option to export drawings to XML format for use with other applications. Exported XML should accurately represent the objects and their properties in the drawing.

Chapter 4

Non-Functional Requirements

4.1 Usability

The program should have an intuitive user interface with clear menus, buttons, and visual feedback. Users should be able to quickly learn and navigate the program's features.

4.2 Performance

Drawing and manipulation operations should be responsive, even with complex drawings. File loading and saving should be efficient, with minimal delay for large drawings.

4.3 Scalability

The program should be able to handle an indefinite number of object types and drawing operations. It should support future extensions, such as new object types and operations, without major redesign.

4.4 Compatibility

The program should run on common operating systems such as Windows, macOS, and Linux. Exported drawings and XML files should be compatible with other applications and platforms.

Chapter 5

System Architecture

5.1 UML Class Diagram and Table

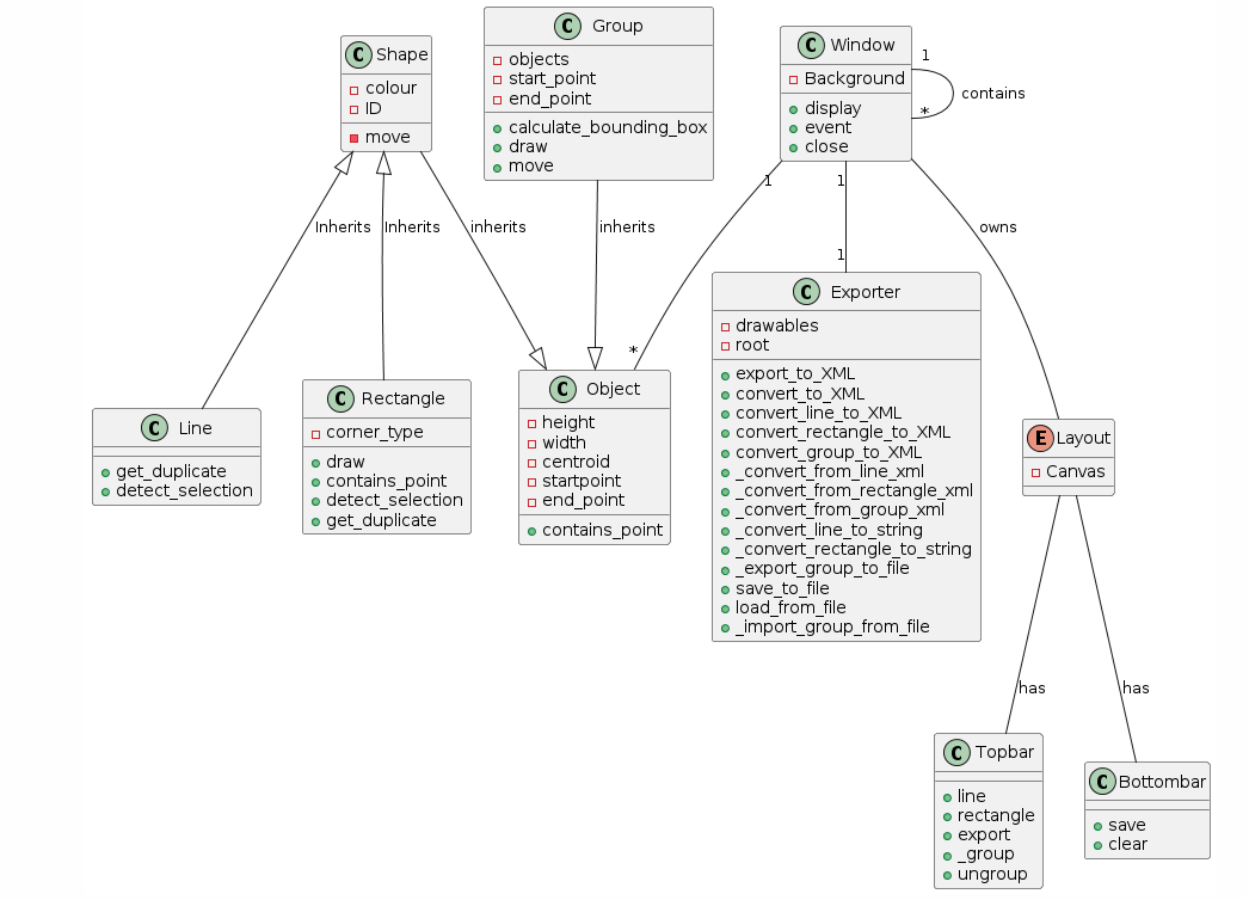
Below we present a Class Table which presents all our classes and summarizes the major responsibilities of each major class:-

Class	**Major Responsibilities**
Window	Manages the main window of the drawing program. Handles background settings, display, events, and closing.
Object	Represents basic drawing objects. Manages properties such as height, width, centroid, start and end points, and provides methods for checking containment.
Topbar	Provides functionalities related to drawing operations, including line and rectangle drawing, export, grouping, and ungrouping.
Bottombar	Handles operations such as saving and clearing the drawing.
Exporter	Manages exporting drawings to XML format. Contains methods for converting drawings, lines, rectangles, and groups to XML.

Table 5.1: Classes and Major Responsibilities

Class	Major Responsibilities
Shape	Represents basic shape properties such as color and ID.
Line	Represents lines in the drawing. Provides methods for drawing, moving, and detecting selection.
Rectangle	Represents rectangles in the drawing. Provides methods for drawing, moving, and detecting selection, as well as handling corner types.
Group	Represents grouped objects. Manages objects within a group, including bounding box calculation, drawing, and moving.
BoundingBox	Represents the bounding box of objects or groups. Manages properties such as height, width, centroid, and orientation, and provides a method for checking containment.
Layout	Represents the canvas layout.

Table 5.2: Classes and Major Responsibilities

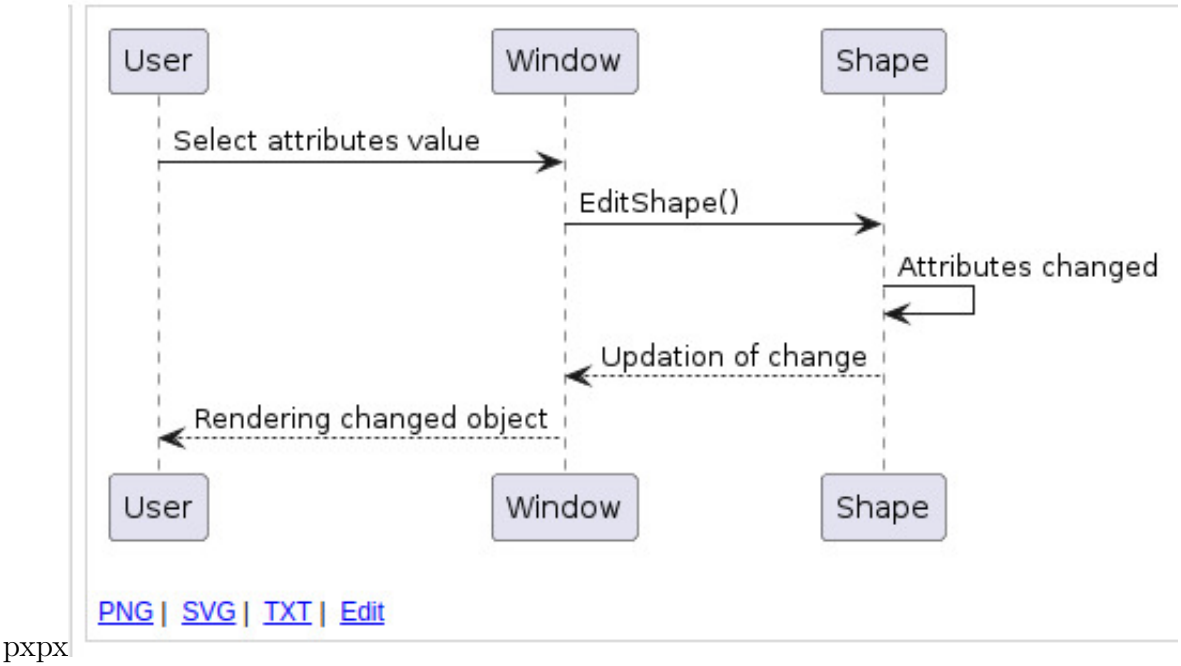


Chapter 6

Sequence Diagram

6.1 Menu choice to edit the current object, user change of attributes, user confirmation of change, update of element in the model, refresh of the drawing

User inputs the attribute values, Window gets this input and passes the change details to Shapes which modifies the current object,the change is made and updated to the window section which finally renders it for the User.

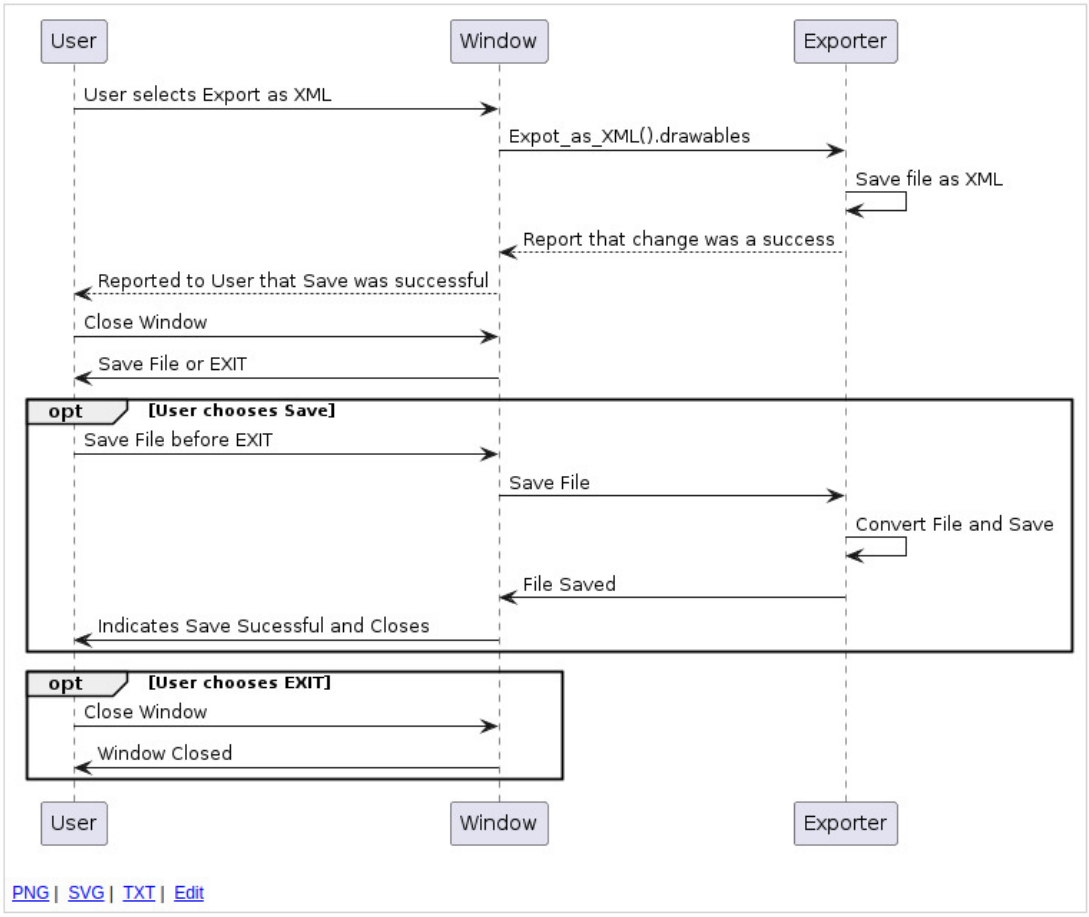


pxpx

Figure 6.1: Edit Sequence

6.2 User indicates to save an XML file, contents of drawing being written to the file, file closed

User inputs that they want to save as an XML file to the Windows which forwards this request to the exporter. Here the exporter converts the file to an XML file as per users request and then reports that the change was successful to the Window, which further reports to user that the change was a success. Now when user tries closing the window they are presented with an option to either save the window or not. If they save, then the above cycle takes place again or else the Window just closes.



pxpx

Figure 6.2: Edit Sequence

Chapter 7

Design Narrative

7.1 Low Coupling and High Cohesion

- The design achieves low coupling by ensuring that classes are loosely connected and interact through well-defined interfaces. Each class is responsible for a specific task or set of tasks, minimizing dependencies between classes.
- High cohesion is achieved by grouping related functionalities within classes. For example, the Object class encapsulates properties and methods related to basic drawing objects, while the Topbar class handles drawing operations.

7.2 Separation of Concerns

- The design separates concerns by assigning distinct responsibilities to different classes. For example, the Window class manages the main window interface, while the Exporter class handles exporting drawings to XML format.
- Each class focuses on a specific aspect of the system, making it easier to understand, maintain, and modify.

7.3 Information Hiding

- Information hiding is implemented by encapsulating internal details within classes and exposing only necessary interfaces to the outside world.
- For example, the Object class hides implementation details such as height, width, and centroid, allowing clients to interact with objects through well-defined methods like contains_point.

7.4 Law of Demeter

- The design adheres to the Law of Demeter by minimizing dependencies between classes and avoiding deep chains of method calls.
- For example, the Topbar class interacts directly with the Window class to perform drawing operations, rather than accessing objects through intermediate classes.

7.5 Extensibility and Reusability

- The design is extensible, allowing for the integration of new object types, operations, editors, and file formats.
- Reusability is promoted through the use of design patterns such as composition and inheritance. For example, the Group class can contain other objects, enabling the creation of nested groups and facilitating code reuse.

7.6 Design Patterns

- The design employs several design patterns to achieve its goals. For example:
  - The **Composite** pattern is used to represent grouped objects, allowing clients to treat individual objects and groups uniformly.
  - The **Strategy** pattern is employed to handle exporting drawings to different file formats. Exporter class encapsulates various strategies for converting drawing elements to XML.

7.7 Expected Product Evolution

- The design anticipates future product evolution by providing flexibility and scalability. New object types, operations, editors, and file formats can be easily integrated without major redesign.
- For example, the Exporter class can accommodate new export formats by adding additional strategies, and the Group class supports arbitrary depth grouping, allowing for complex structures in future versions.

Bibliography

Tools Used:

1. **Overleaf** [for LaTeX formatting and Documentation]:
    - Overleaf is an online platform for collaborative LaTeX editing and document preparation. It provides a user-friendly environment for writing, editing, and compiling LaTeX documents in real-time, with features such as automatic compilation, version control, and sharing options. Overleaf simplifies the process of creating professional-looking documents, including research papers, reports, and presentations, making it popular among researchers, academics, and students.
2. **PlantUML** [for classs and sequence diagrams]:
    - PlantUML is an open-source tool for creating UML diagrams using a simple and intuitive textual syntax. It supports various types of diagrams, including class diagrams, sequence diagrams, use case diagrams, and more. PlantUML generates diagrams automatically from textual descriptions, making it easy to create and maintain visual representations of system designs. Its flexibility and ease of use make it a preferred choice for developers and designers who prefer a text-based approach to diagramming.
3. **GitHub** [for Version control and Collaboation]:
    - GitHub is an online software development platform. It's used for storing, tracking, and collaborating on software projects. It makes it easy for developers to share code files and collaborate with fellow developers on open-source projects.