Dr. UML

May 19, 2025

CSIE IV 蕭耕宏110590005
CSIE IV 黃冠鈞110590028
CSIE IV 張庭瑋110590035
CSIE IV 吳宥駒110590066
Homework #6

1 Change History

1.1 HW1

- Add section Problem statement.
- Add section Development Language.

1.2 HW2

- Add section 3-8.
- Change section "Development Language" to "Software Environments" as demanded.
- Change the project name to "Dr. UML".

1.3 HW3

- Add section 9(Domain Model).
- Merge UC7(Host Session) and UC8(Join Session).
- Change Acronyms for System Features into FEA.

1.4 HW4

- Extract UC01-extension *b to UC09.
- Refine pre-condition of UC01.
- Use case
  - Remove extension "3.a If User connects Gadget to nothing, a default Gadget will be generated automatically." in UC01.
- Domain Model
  - Add Timer and Verifier and their associates.
  - Add zIndex attribute to Components.
- Add Logical architecture
- Add System Sequence Diagrams with GRASP Patterns
- Add Design Class Model

1.5 HW6

- Update Figure
  - Domain model with associations
  - Domain model with associations and attributes
- Use case
  - UC01
    - In Main Success Scenario step 1, Change the wording from "drag" to "select".
    - Replace "1-4 steps can be repeated" in the Main Success Scenario with "User can skip any step from 1 to 4" in the Extension.
    - Remove extension 4.d.
- Domain Model
  - Remove TextFiled, ImageFormat, Filename, Field, and Component from Bad

2 Problem statement

There are several tools available for creating UML diagrams on the internet but many of them come with paid subscriptions or limitations that make them less accessible. Moreover, they often resort to creating poorly formatted documents due to the lack of affordable, high-quality options. As the result, we propose Dr. UML.

Dr. UML is an innovative collaborative platform designed for software developers, system architects, and students who need to efficiently create and manage Unified Modeling Language (UML) diagrams.

The tool meets the pressing need for collaborative designing by allowing teams to work together simultaneously, regardless of their physical location. In addition, it offers real-time updates and integrated communication features. Dr. UML will be used primarily in design meetings, brainstorming sessions, and technical workshops where immediate visual feedback is essential. It is needed when precise and dynamic visual representation of complex systems is required to align team understanding and streamline development processes.

Dr. UML integrates a robust set of customizable UML elements with drag-and-drop functionality and real-time collaboration. This not only enhances the creative aspects of system design but also ensures that technical requirements are met with precision and clarity. The platform's intuitive design and collaborative capabilities make it an essential tool for modern software development teams, system architects, and students aiming to create high-quality UML diagrams efficiently.

3 Summary of System Features

- FEA01: Create a UML diagram file.
- FEA02: Edit a UML diagram(draw, edit Component properties, copy and paste Components)
- FEA03: Save and load progress.
- FEA04: Export UML diagram into image formats.
- FEA05: Start a online Session, allowing other Users to join.
- FEA06: Connect to online Sessions, edit UML with other users simultaneously.
- FEA07: Real-time chatroom in a online Session.

4 Use Case Diagram

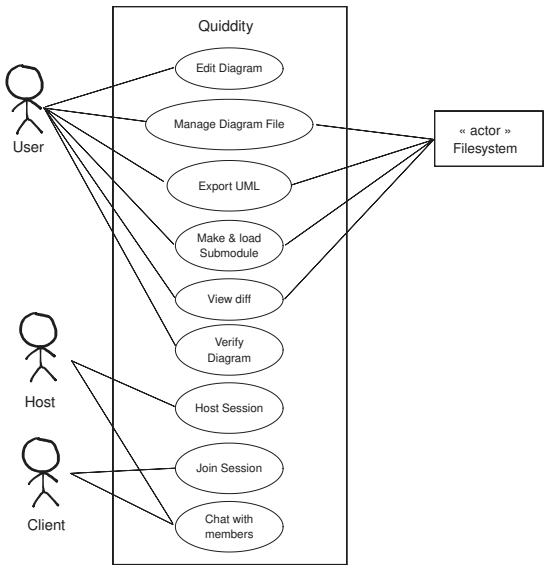


Figure 1: Use case diagram

5 Use Cases

5.1 UC01: Edit UML

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User

- **Stakeholders and Interests**:
  - **User**: Wants to create and connect Components.
- **Preconditions**:
  - User has opened a UML Project.
  - At least one UML Diagram is opened.
  - UML Editing Canvas and Toolbox are loaded.
- **Success Guarantee**: UML is edited according to the User's specifications.
- **Main Success Scenario**:
  1. User selects Gadgets from Toolbox.
  2. User edits the Gadgets.
  3. User establishes connections between Gadgets via Associations.
  4. User modifies the Associations as needed.
- **Extensions**:
  - *a In the event of System failure, User restarts System. UML will revert to the last successfully saved state.
  - *b When editing text, User may design their text as desired(See UC09)
  - *c User may select Gadgets within the canvas.
  - *d When User drags Gadget with multiple Associations, System will automatically update them.
  - *e User can determine the layering order when Gadgets overlap.
  - *f User may copy and paste Components.
    1. Copying or pasting Associations will also include connected Gadgets.
  - *g User may undo or redo actions.
  - *h User can skip any step from 1 to 4
  - 1.a Edit fails if Gadget is dragged to an invalid location.
  - 1.b User can also import an available Submodule in the current Project.
  - 2.a Different types of Gadgets will have distinct Fields available for editing.
  - 2.b Edited Gadgets will automatically scale to fit the changes.
  - 2.c User can modify the color of a Gadget. The color will apply to Gadget's background.
  - 2.d User can move the Gadget as long as the destination is valid. The associations will be updated automatically.
  - 3.a Deleting a Gadget will also remove the associated connections.
  - 3.b Self-Associations are allowed.

  - 4.a User may change the type of Association.
  - 4.b User may add, remove, and move text Fields in Association.
  - 4.d User can modify the path of an Association.

- **Special Requirements**:
  - When multiple users are editing, no unintended behavior should occur.
- **Frequency of Occurrence**: Often
- **Open Issues**: None specified

5.2 UC02: Manage UML

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User
- **Stakeholders and Interests**:
  - **User**: Wants to create, save, export, delete a Diagram.
  - **Filesystem**: Requires the file to be read/saved properly.
- **Preconditions**:
  - System is up.
- **Success Guarantee**:
  - User manage Project and Diagram.
- **Main Success Scenario**:
  1. User selects existing Project.
  2. User selects Diagram in the Project.
  3. User edit Diagram as described in UC1.
  4. User exports Diagram to Filesystem.
  5. User deletes the Diagram.
- **Extensions**:
  - *a In both Project and Diagram, the time of last edit is recorded and can be viewed by User.
  - *b If System fails manage (load, save, and export) Project or Diagram, User will be prompted to either retry the operation or abort.
  - 1.a User may choose to create a new Project.

  - 1.b User can modify the name of the Project.
  - 2.a User may choose to create a new Diagram.
  - 3.a User can modify the type, background color, filename of the Diagram.
  - 4.b When exporting Diagram, User may select one of the supported UML formats (for future verification purposes).
- **Frequency of Occurrence**: Occasionally
- **Open Issues**: None specified

5.3 UC03: Export UML

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User
- **Stakeholders and Interests**:
  - **User**: Wants to export UML-Project to image formats with desired name and extension.
  - **Filesystem**: Requires the exported image to be saved properly.
- **Preconditions**:
  - System is up.
  - User has opened a UML project.
- **Success Guarantee**: Exported image is saved to Filesystem.
- **Main Success Scenario**:
  1. User starts the exporting current UML.
  2. User selects a supported image format and specifies a filename.
  3. System saves the exported image to Filesystem.
- **Extensions**:
  - 2.a Supported formats:
    * JPEG
    * PNG
    * SVG
    * WebP
  - 3.a If Filesystem fails to save the exported image, User will be prompted to either retry the operation or abort.
- **Frequency of Occurrence**: Occasionally
- **Open Issues**: None specified

5.4 UC04: Manage Submodule

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User
- **Stakeholders and Interests**:
  - **User**
    - Imports Submodule to UML Project, which can be included it to any UML Diagram in the Project.
    - Removes any imported Submodule(s) from UML Project
    - Exports a part of components to a Submodule and save it to Filesystem.
  - **Filesystem**: Wants to save and load a Submodule file.
- **Preconditions**:
  - System is up.
  - User has opened a UML-Project.
  - For exporting Submodule, User is also required to open a UML Diagram and have a number of Component drawn.
- **Success Guarantee**:
  - Submodule is exported on Filesystem.
  - Submodule is imported to UML-Project.
- **Main Success Scenario**:
  1. User imports a Submodule file from Filesystem.
  2. From the Project menu, User select a outdated Submodule and remove it.
  3. After drawing a number of Component, User selects and export them as a Submodule.
  4. The Submodule file is saved to Filesystem.
- **Extensions**:
  - 1.a If Filesystem fails to import Submodule, User will be prompted to either retry the operation or abort.
  - 3.a If nothing is selected, User cannot export it as Submodule.
  - 3.b Optionally, the saved Submodule may have empty Fields.
  - 4.a If Filesystem fails to save Submodule file, User will be prompted to either retry the operation or abort.
- **Frequency of Occurrence**: Sometimes
- **Open Issues**: None specified

5.5 UC05: Verify UML

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User
- **Stakeholders and Interests**:
  - **User**: Wants to verify the correctness of UML.
- **Preconditions**:
  - System is up.
  - A UML is available.
- **Success Guarantee**: System verifies and displays the correctness of the UML diagram.
- **Main Success Scenario**:
  1. User opens a UML project.
  2. User instructs System to verify the UML.
  3. System checks the correctness of the UML.
  4. System displays the verification results.
- **Extensions**:
  - 1.a If System fails to open the UML file, User will be prompted to either retry the operation or abort.
  - 3.a If System fails to verify the UML, User will be prompted to either retry the operation or abort.
  - 3.b System verifies the UML by diagram type.
    1. If the UML type verification is unavailable, inform the User.
  - 4.a If System fails to display result, User will be prompted to either retry the operation or abort.
  - 4.b System informs User of verification results, which may include:
    1. All clear, UML is correct in terms of diagram type.
    2. Warnings, UML is mostly free of syntax errors, but contains some bad smells™.
    3. Invalid, UML contains critical errors.
- **Special Requirements**:
  - Error messages should be user-friendly and provide actionable insights.
- **Technology**:
- **Frequency of Occurrence**: Sometimes
- **Open Issues**: None specified

5.6 UC06: Join Session

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: Client
- **Stakeholders and Interests**:
  - **Host**: Wants Client to join current opened project.
  - **Client**: Wants to connect to an existing project.
- **Preconditions**:
  - System is up.
  - A stable connection exists between Host and Client.
  - Host opened a UML project.
- **Success Guarantee**: Connection is established and maintained, UML is synced, and no conflicts occur.
- **Main Success Scenario**:
  1. Host starts a Session.
  2. Client connects to the Session.
  3. Both Host and Client edit the UML as described in UC1.
  4. Step 3 is repeated until UML is complete.
  5. Host ends the Session.
- **Extensions**:
  - *a User may opens a session for a project. Assuming the role of Host.
  - *b At any time, a Component can only be edited by exactly one User.
  - 2.a If a connection error occurs, notify Client of the issue encountered.
  - 3-4.a Host can remove any Client from the Session.
  - 3-4.b If an action fails to send, the System retries the action.
    1. If retry limit is reached, System will remove that Client from current Session.
  - 3-4.c Clients may redo and undo their own edits.
  - 5.a Session ends if Host closes it, whether intentionally or unintentionally.
- **Frequency of Occurrence**: Sometimes
- **Open Issues**: Undo/redo conflicts

5.7 UC07: Chat with Members

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User
- **Stakeholders and Interests**:
  - **User**: Wants to communicate with other users in Session via text messages.
- **Preconditions**:
  - System is up.
  - Users have joined a Session.
- **Success Guarantee**:
  - Users in Session can communicate with each other with text messages.
  - Users in Session can view chat history.
- **Main Success Scenario**:
  1. User opens the chatroom for current Session.
  2. User views chat history.
  3. User types and sends messages.
- **Extensions**:
  - *a The time of each sent message is recorded and can be viewed by User.
  - 1.a If a new Session is created, System creates a new chatroom with no messages.
  - 2.a If System fails to load the chatroom, it will attempt to retry.
    1. If retry limit is reached, notify User of the issue encountered. User will not be able to view the previous messages.
  - 3.a If System fails to send User's message, it prompts the User to either remove or resend the message.
- **Frequency of Occurrence**: Sometimes
- **Open Issues**: None specified

5.8 UC08: Edit Attribute of a Component

- **Scope**: Dr. UML
- **Level**: User goal
- **Primary Actor**: User
- **Stakeholders and Interests**:
  - **User**: Wants to edit Attributes of a existed Component.
- **Preconditions**:
  - A Component is created.
- **Success Guarantee**:
  - Attributes is edited according to the User's specifications.
- **Main Success Scenario**:
  1. User select a Component.
  2. User add a new Attribute into the Component.
  3. User select an Attribute.
  4. User change the size, font, type of the texts.
  5. System updates according to last step.
  6. Steps 1–5 are repeated in any order until the editing is complete.
- **Extensions**:
  - *a If a Component or Attribute is deselected during editing, System saves its current state.
  - 2.a User may select an exited Attribute instead of creating a new one.
  - 3.a After an Attribute is selected, User may also remove it from the Component.
  - 4.a Text type has these options:
    - Bold
    - Italic
    - Underline
- **Frequency of Occurrence**: Often
- **Open Issues**: None specified

6 Non-functional Requirements and Constraints

6.1 Performance Requirements

- **NFR1: Response Time**: Operations (e.g., dragging components, editing text, collaboration) should respond within **1s**.
- **NFR2: Concurrent Users**: The system should support at least **4 users** editing a UML diagram in real-time.
- **NFR3: File Handling**: Loading or saving UML files should take **less than 3 seconds** (for diagrams with 100+ elements).
- **NFR4: Export Speed**: UML diagrams should be converted to PNG, JPEG, SVG, or WebP formats within **5 seconds**.
- **NFR5: Network Efficiency**: Collaboration mode should minimize data traffic and prioritize critical updates to reduce bandwidth usage.

6.2 Usability Requirements

- **UR1: User-friendly Interface**: Users should be able to understand basic operations within **20 minutes**.
- **UR2: Undo/Redo**: The system should support **at least 50 levels** of undo and redo history.
- **UR3: Accessibility**: Keyboard shortcuts should be provided to enhance usability.
- **UR4: Collaboration Features**: Users should see real-time updates and be able to communicate via chat or annotations.

6.3 Reliability & Availability Requirements

- **RAR1: Uptime**: The system should maintain **99.9% availability**.
- **RAR2: Autosave**: Progress should be automatically saved every **30 seconds**.
- **RAR3: Error Handling**: The system should handle network failures gracefully, allowing users to reconnect without data loss.
- **RAR4: Data Consistency**: All users should see the same UML diagram state in collaborative mode.

7 Glossary

- Submodule: a part of UML diagram, it can be imported into other UML diagram
- Gadget: a block contains text Fields

- Toolbox: A bar containing Components, allowing Users to add them to the canvas.
- Association: connection between two Gadgets
- Field: the place where User can insert text
- Session: A shared project allowing other Users to collaborate.
- Component: Gadget or Association o the canvas.
- Host: A User who has started a Session.
- Client: A User who has joined a Session.
- User: A general term that refers to any participant, including both the Host and Client.
- Attribute: Property of a Components.
- Attribute Tree: A tree-like UI listing Attributes of a component.

8 Software Environments

Golang.

9 Domain model

9.1 Domain Class Diagram Showing Only Concepts

9.1.1 Classes Identified

The nouns listed below are found in the use case.

Table 1: Classes Identified

*User	System	*UMLDiagram	Component	*Gadget
*Association	TextField	Path	UMLFile	Filesystem
*UMLProject	ImageFormat	Filename	*Submodule	Field
DiagramType	VerificationResult	Host	Client	*Session
Project	Connection	*Chatroom	*Message	ChatHistory
TextStyle				

Note: Classes marked with asterisk(*) are good classes.

9.1.2 Bad Classes

Table 2: Categorization of Terms

Attributes	Abstract Concepts	Implementation Construction	Too UI
fontSize	System	UMLFile	Toolbox
isBold	Host	Filesystem	Canvas
isItalic	Client	VerificationResult	
isUnderline	Project	ChatHistory	
DiagramType		Connection	TextStyle
position			

9.1.3 Good Classes

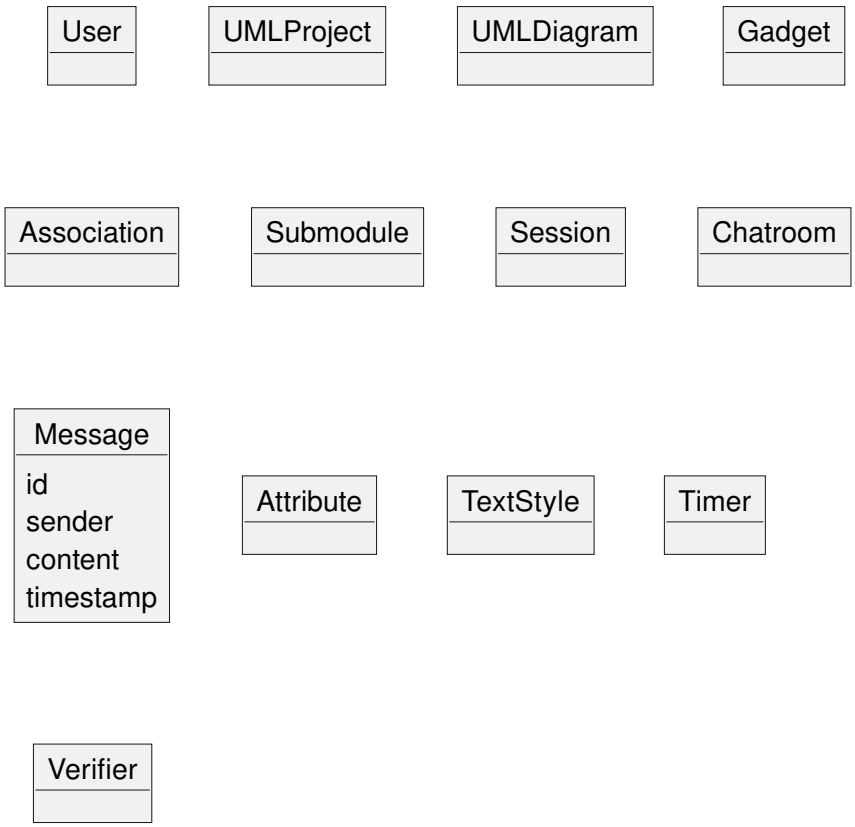


Figure 2: Domain class diagram showing only concepts

9.2 Add Associations

- One User hosts or joins several Sessions.
- One User manages one UMLProject.
- One UMLProject manages one UMLDiagram.

- One UMLDiagram consists of several Submodules.
- One Submodule consists of several Gadgets.
- One Gadget associates with one or two Associations.
- One Association contains several Attributes.
- One Attribute is described by one TextStyle.
- One Session connects to one Chatroom.
- One Chatroom contains several Messages.
- One User sends several Messages.

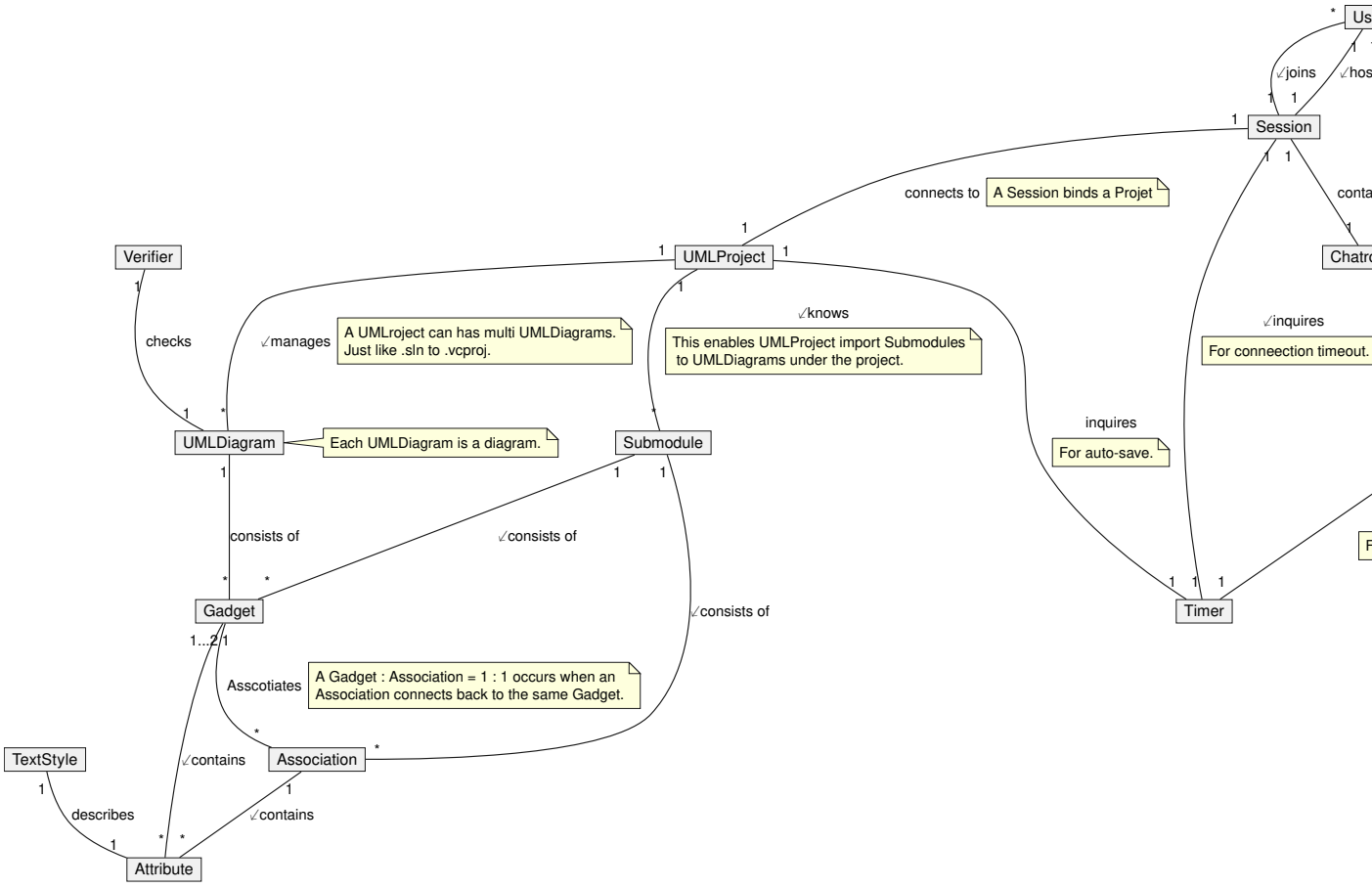


Figure 3: Domain class diagram with associations added

9.3 Add Attributes

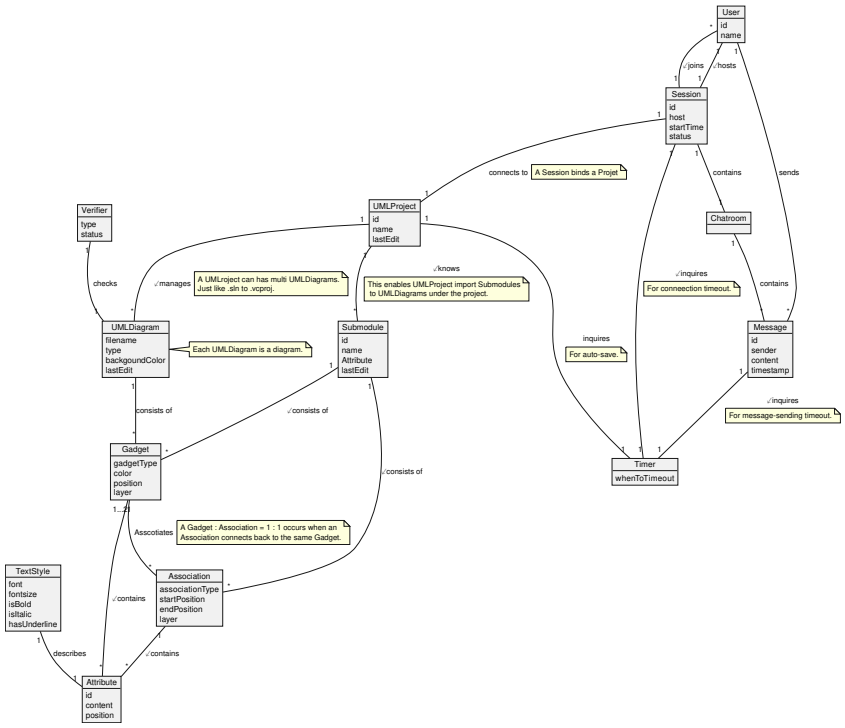


Figure 4: Domain class diagram with attributes added.

10 Logical Architecture

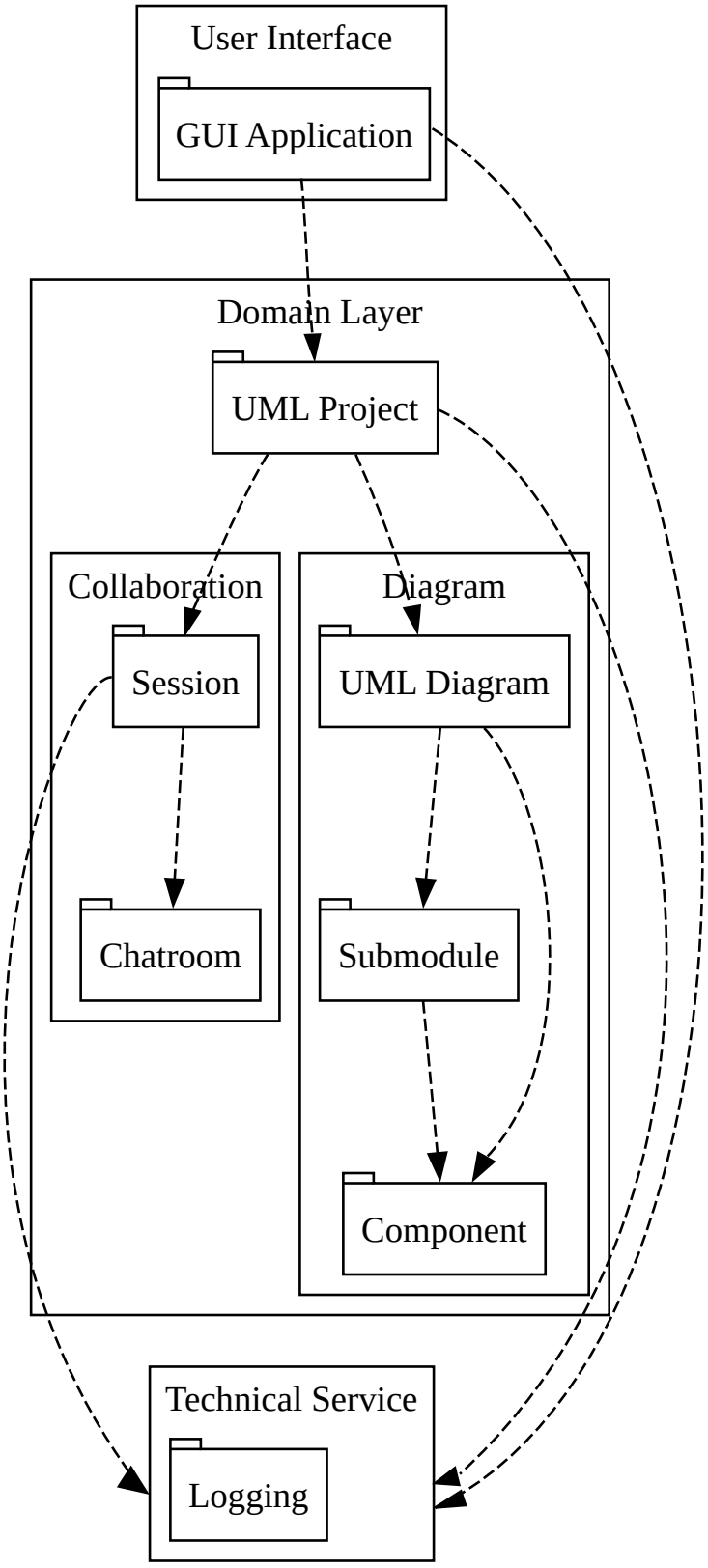


Figure 5: Logical architecture

11 System Sequence Diagrams with GRASP Patterns

We pick UC1 as our most significant use-case. Since UC1 is made of series of system events, we decide to provide SSDs for each one of them.

11.1 main SSD

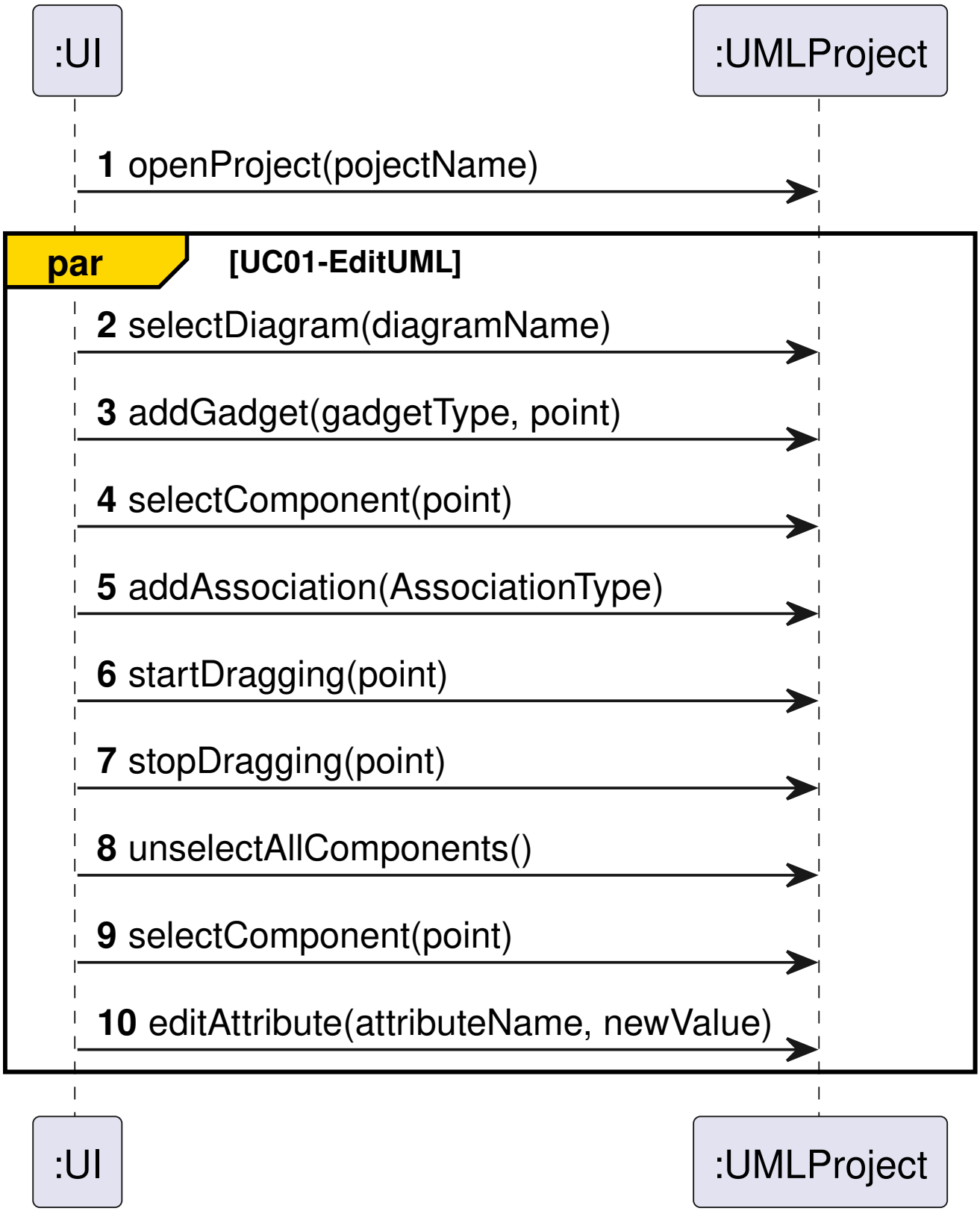


Figure 6: main SSD

11.2 addAssociationToDiagram

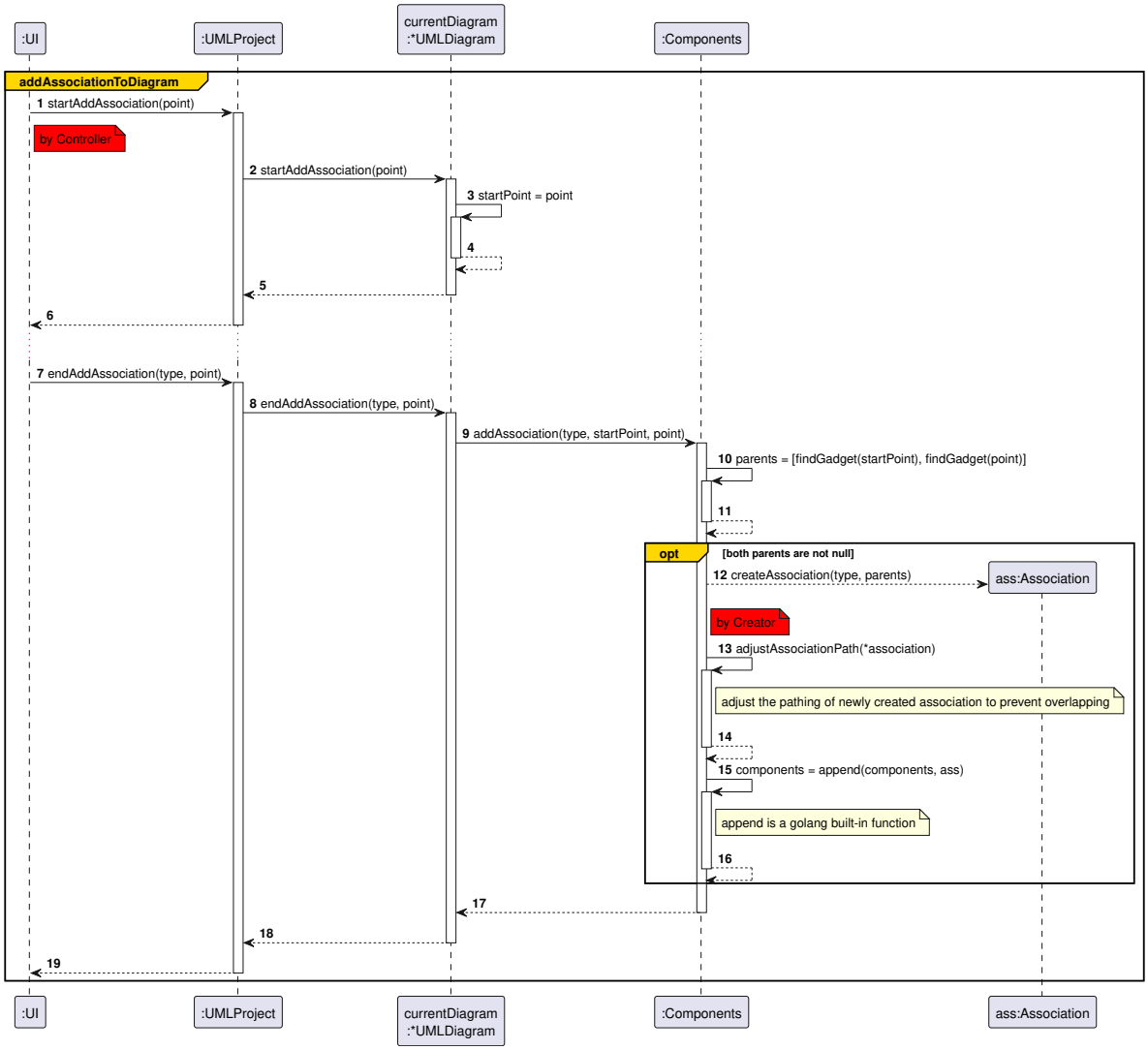


Figure 7: addAssociationToDiagram

tt

11.3 addGadgetToDiagram

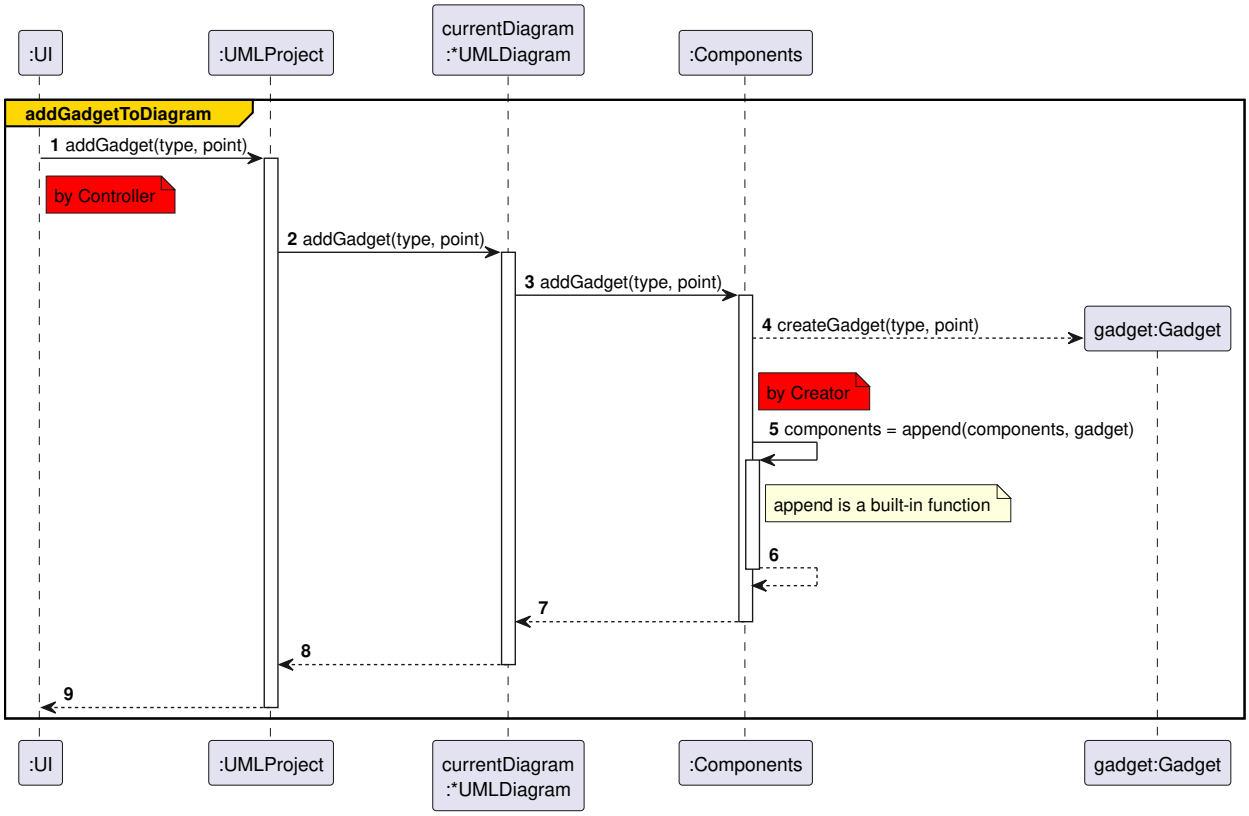


Figure 8: addGadgetToDiagram

11.4 copyComponents

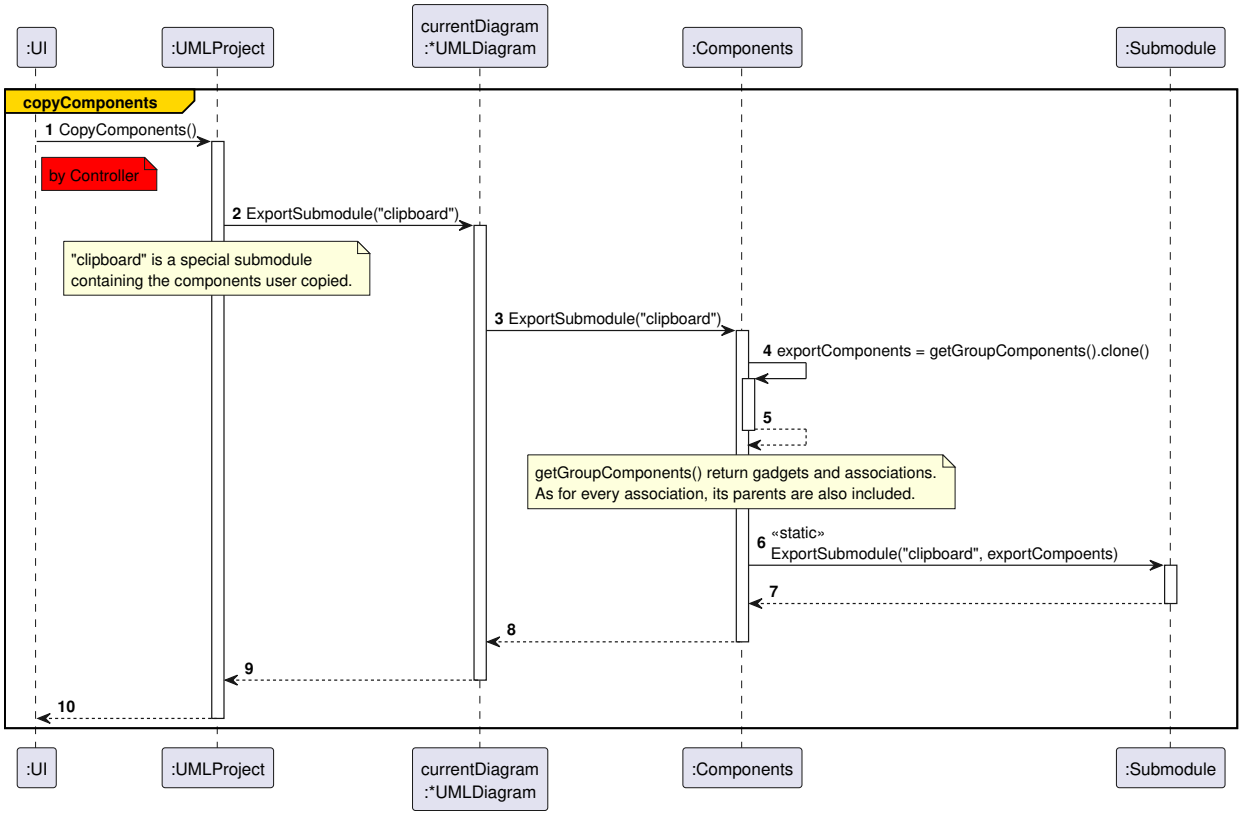


Figure 9: copyComponents

11.5 deleteComponent

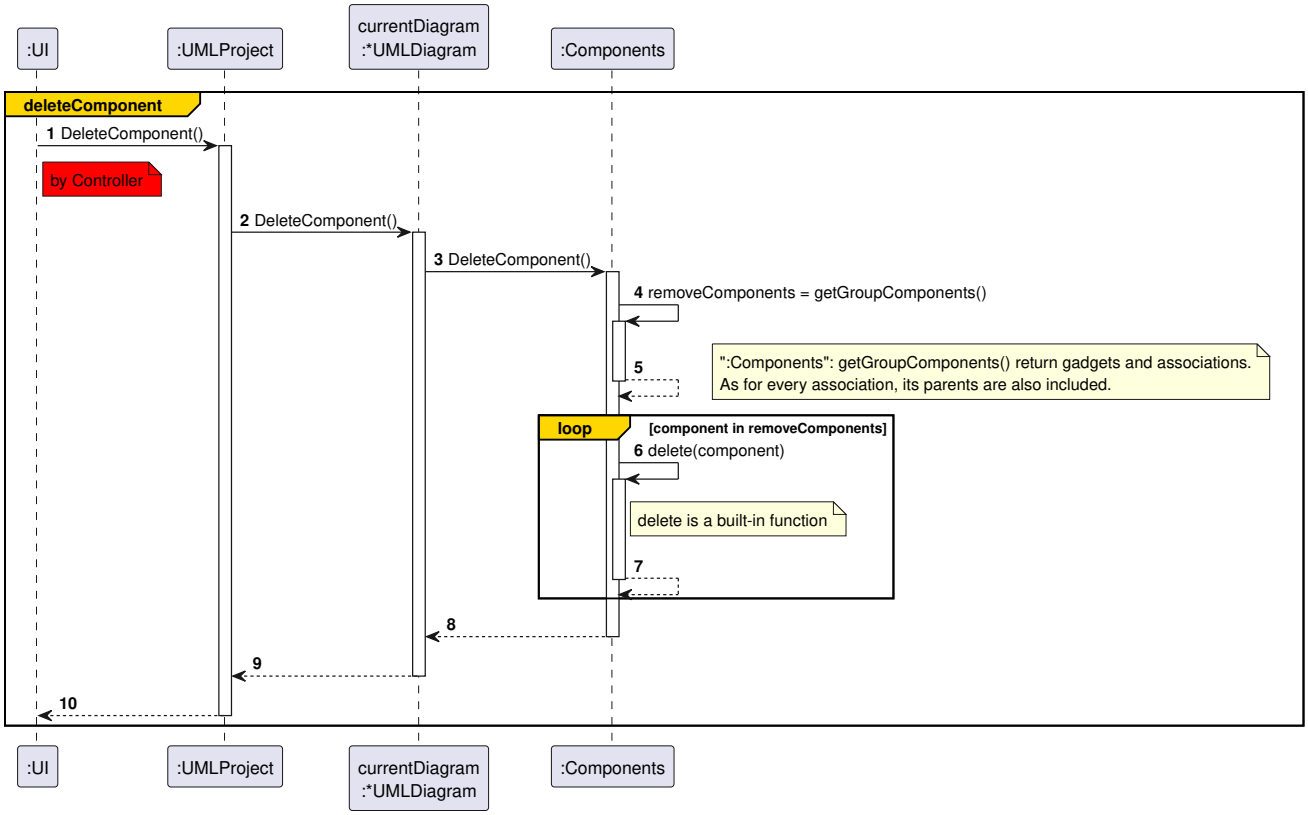


Figure 10: deleteComponent

11.6 drawAll

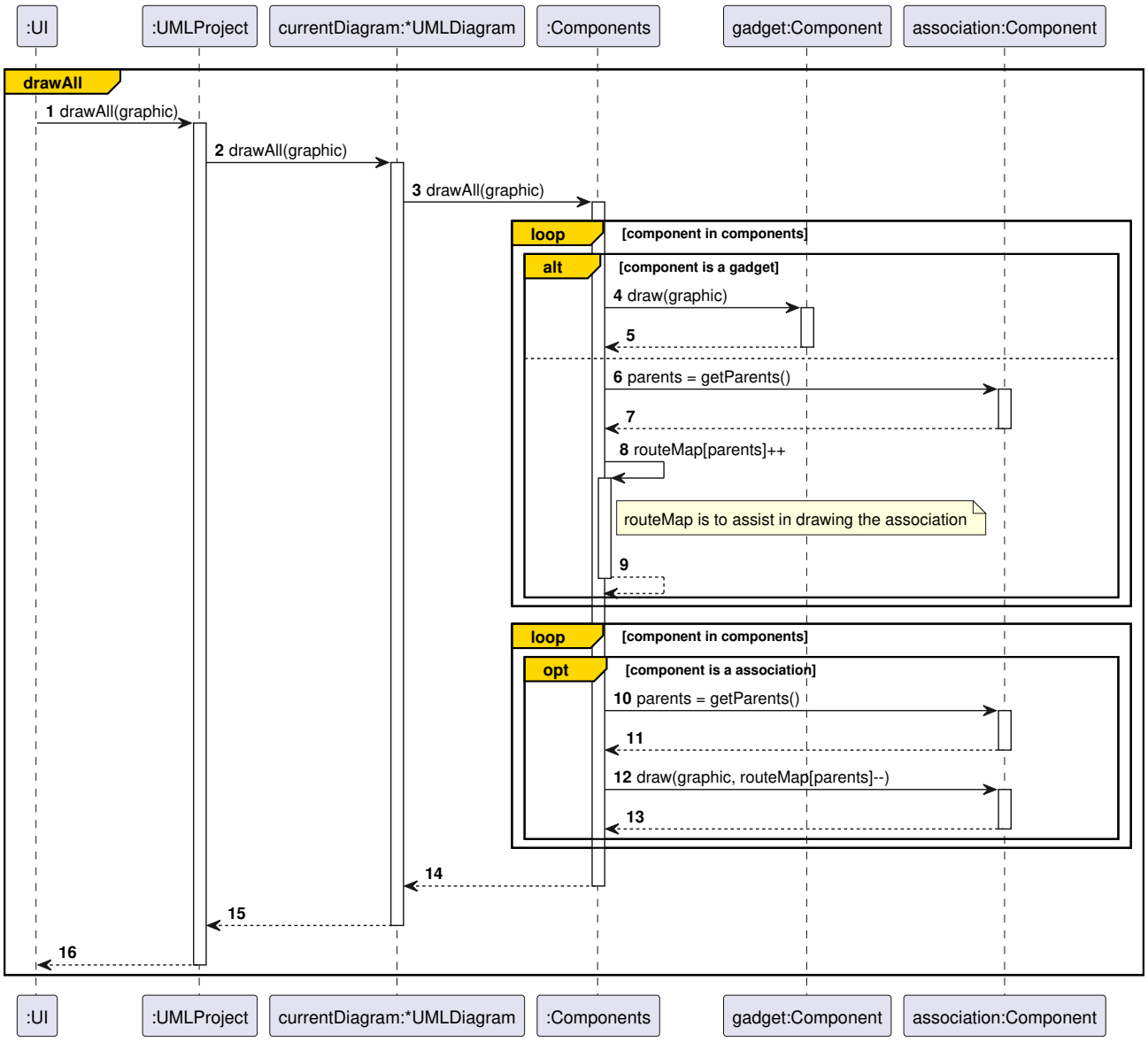


Figure 11: drawAll

11.7 importSubmodule

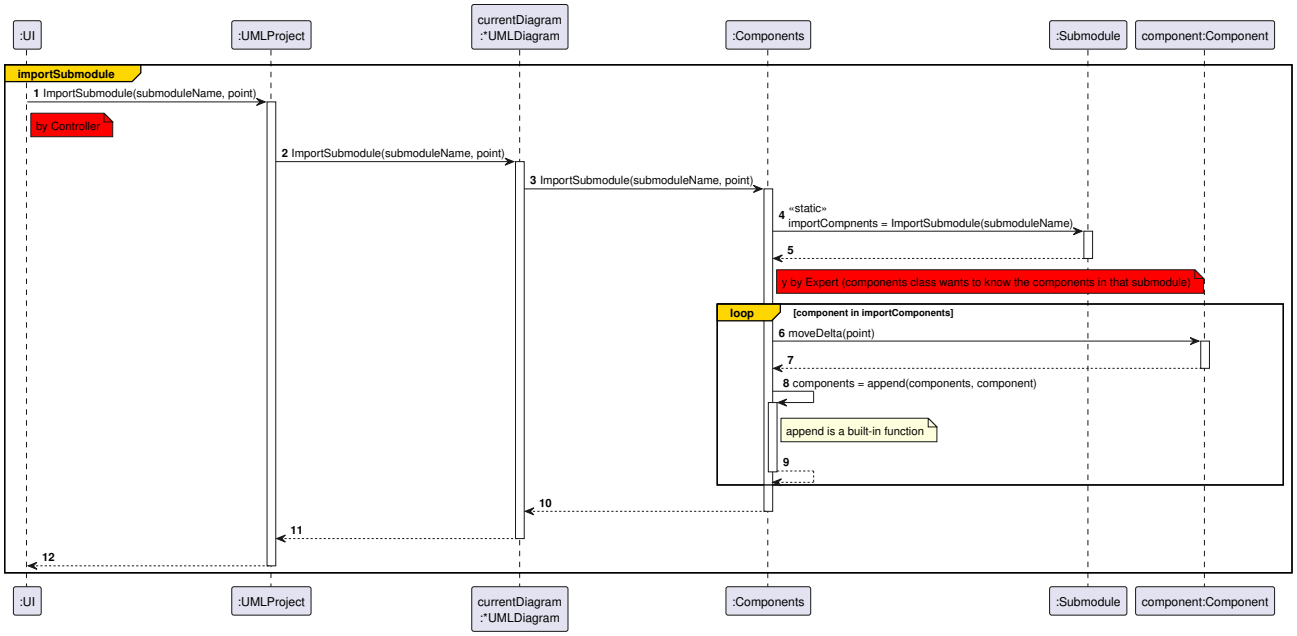


Figure 12: importSubmodule

11.8 moveComponent

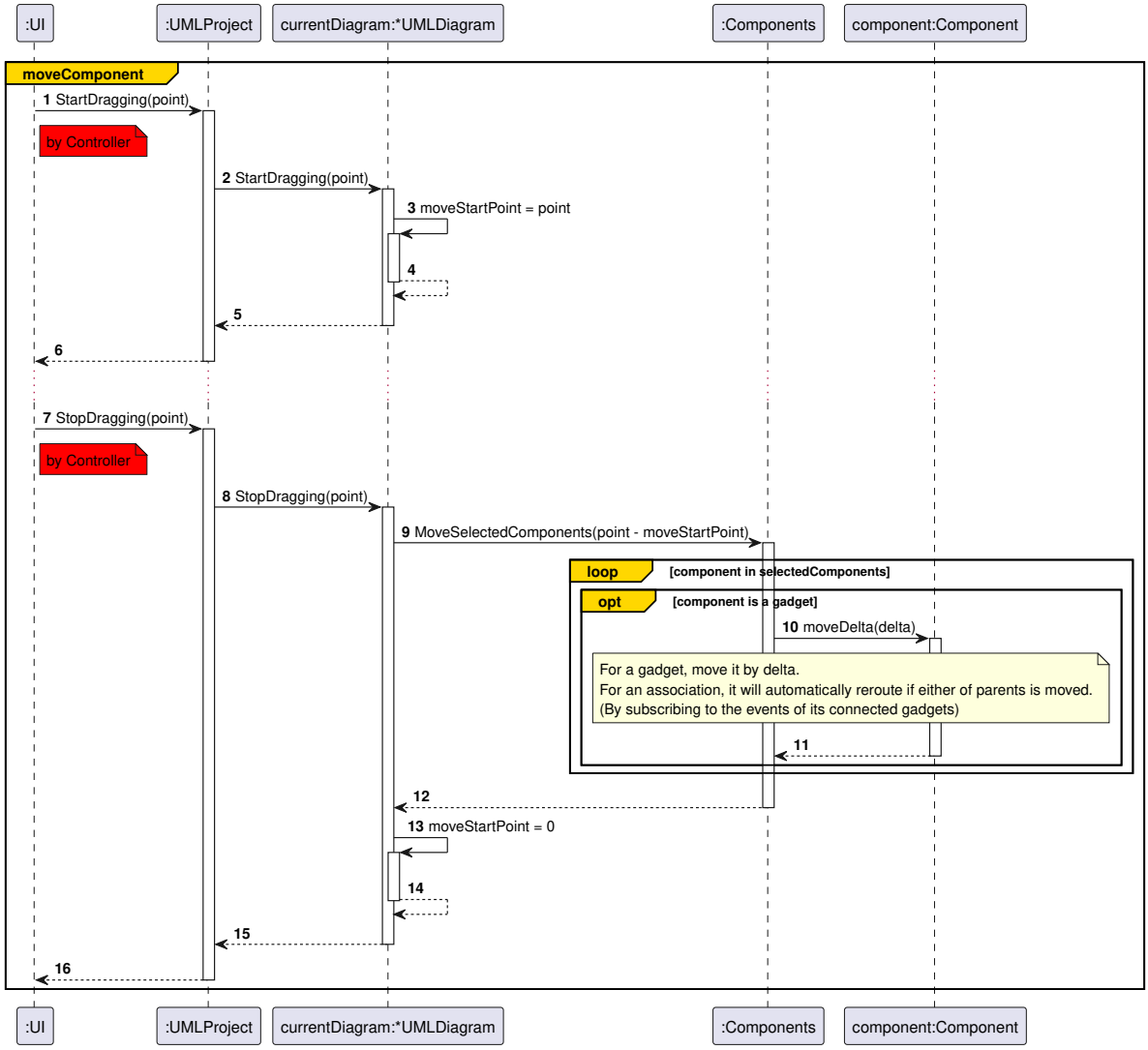


Figure 13: moveComponent

11.9 openProject

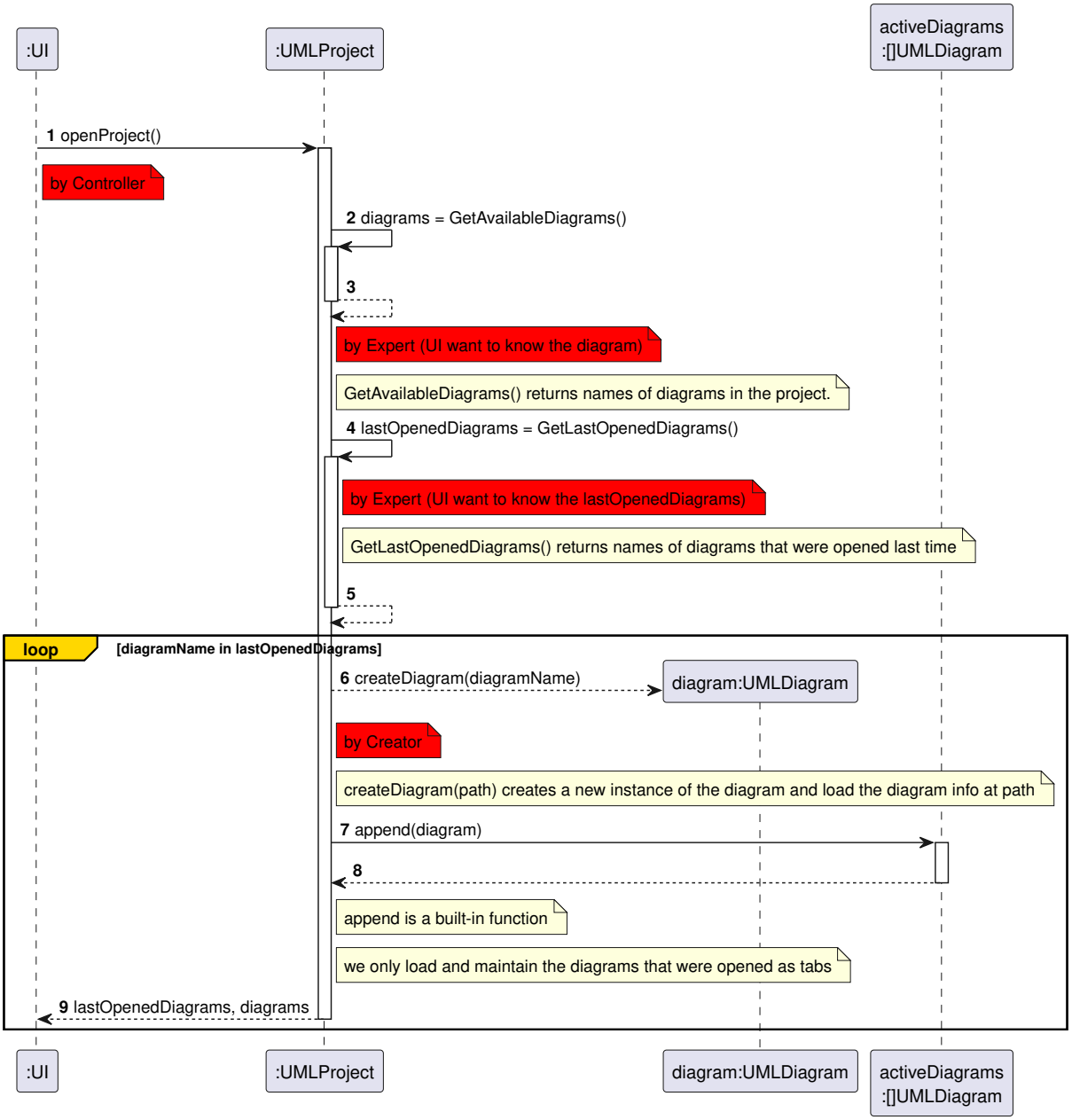


Figure 14: openProject

11.10 pasteComponents

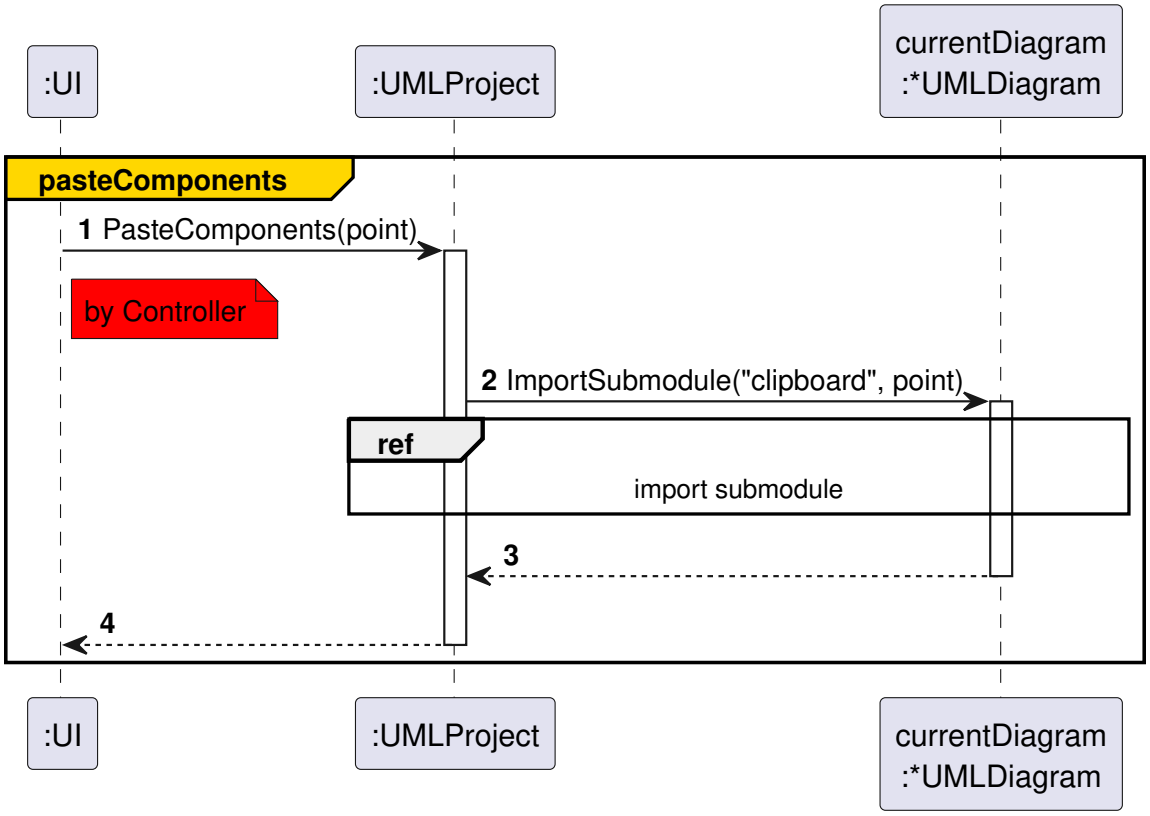


Figure 15: pasteComponents

11.11 redo

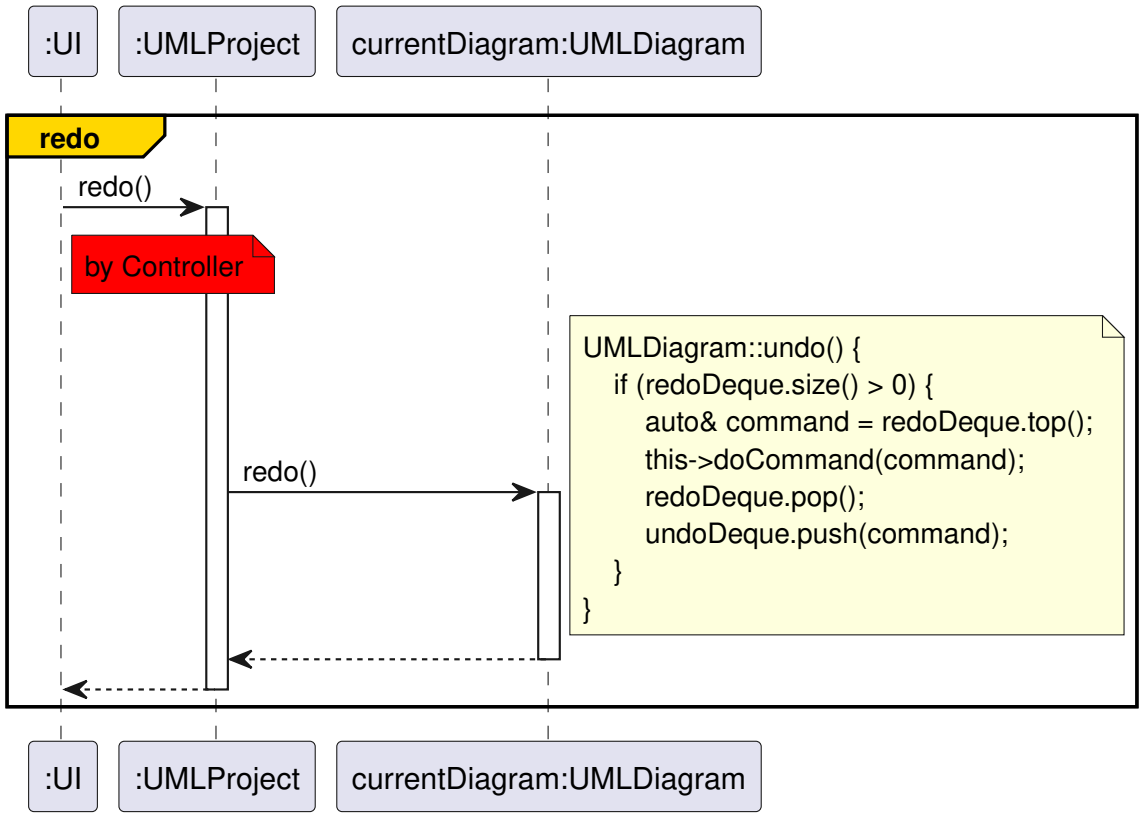


Figure 16: redo

11.12 undo

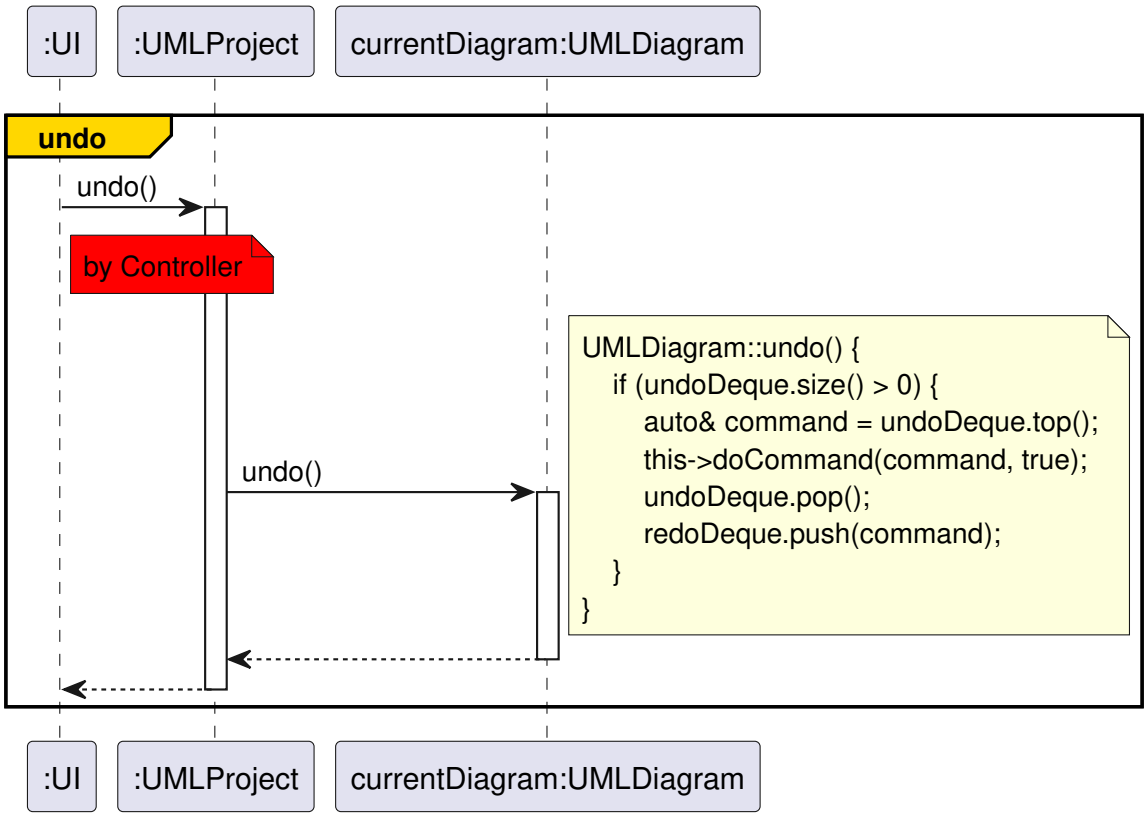


Figure 17: undo

11.13 selectComponent

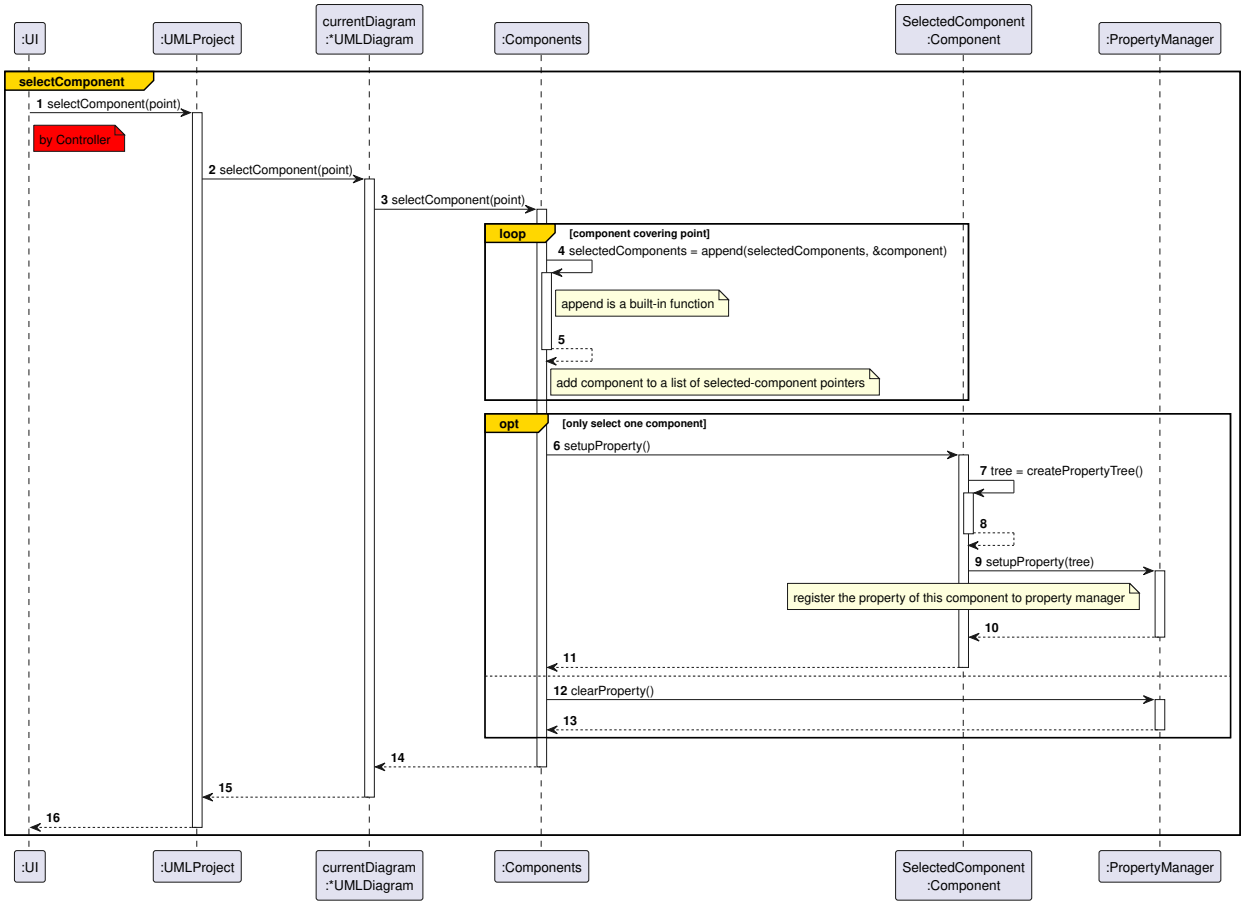


Figure 18: selectComponent

11.14 selectDiagram

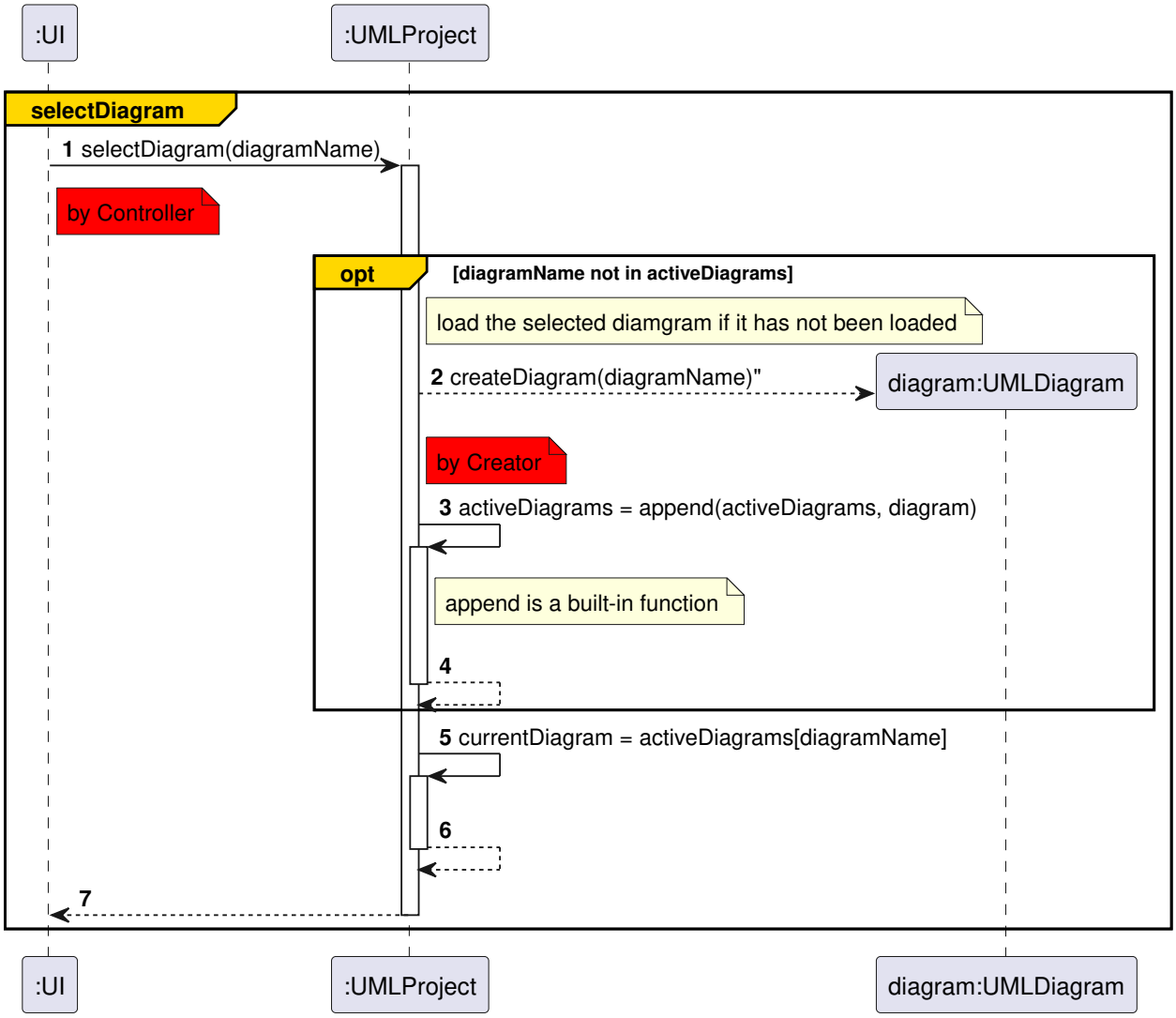


Figure 19: selectDiagram

11.15 unselectAllComponents

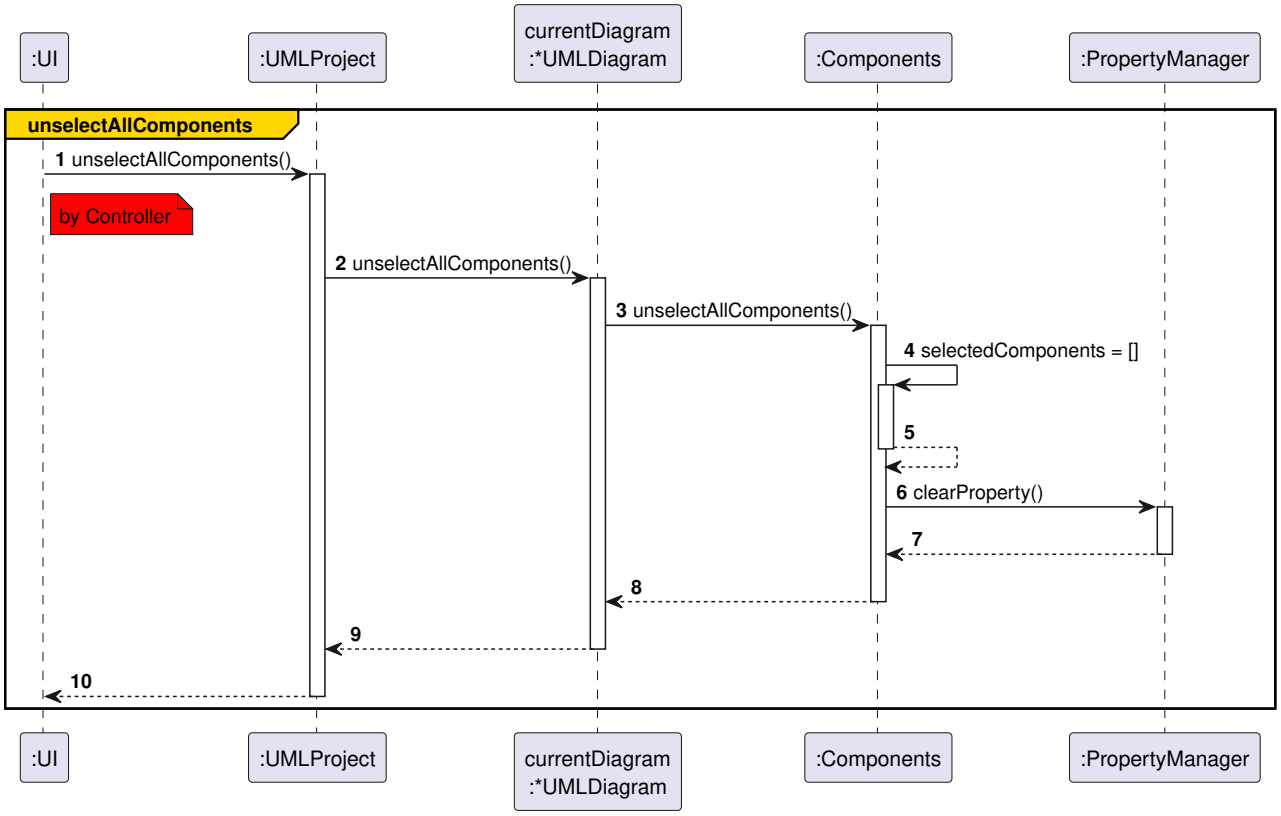


Figure 20: unselectAllComponents

11.16 unselectComponent

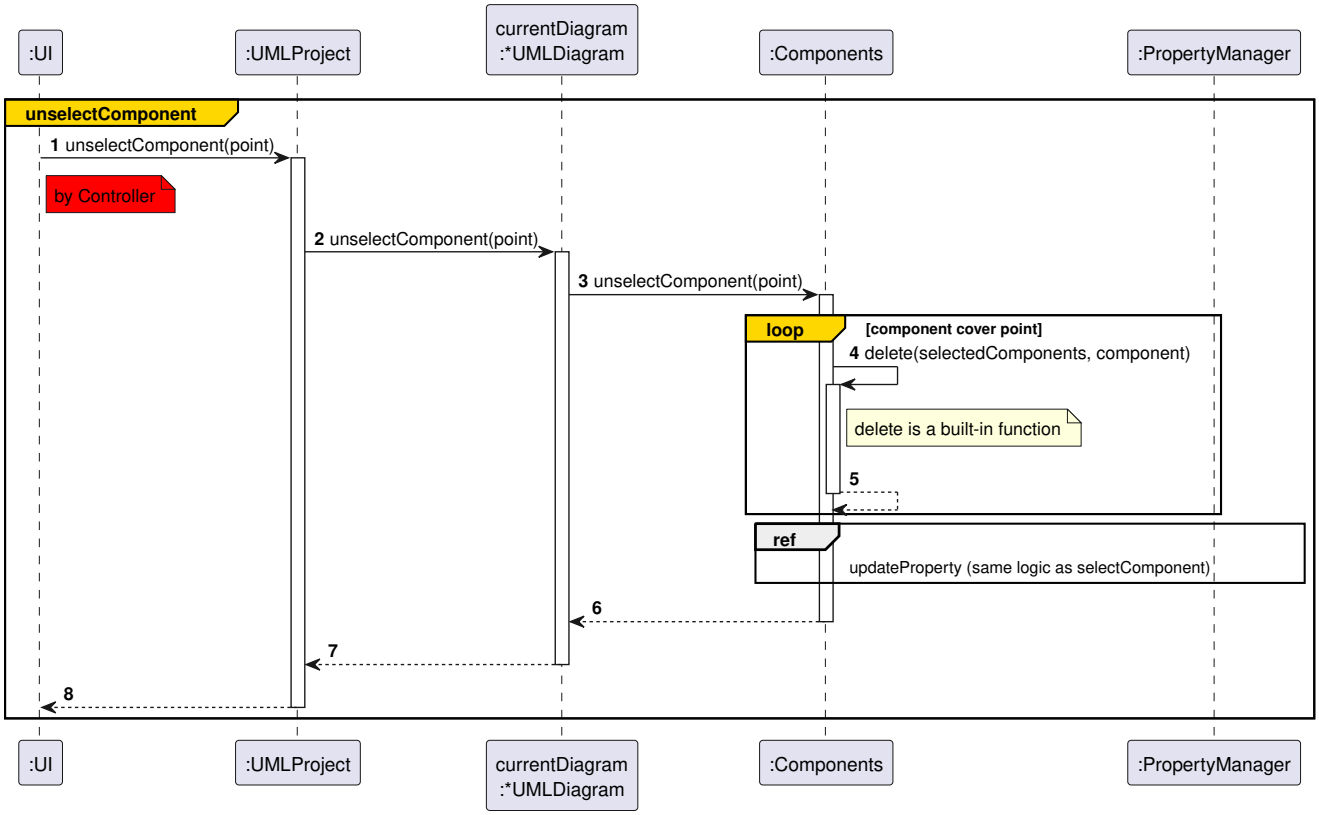


Figure 21: unselectComponent

11.17 updateProperty

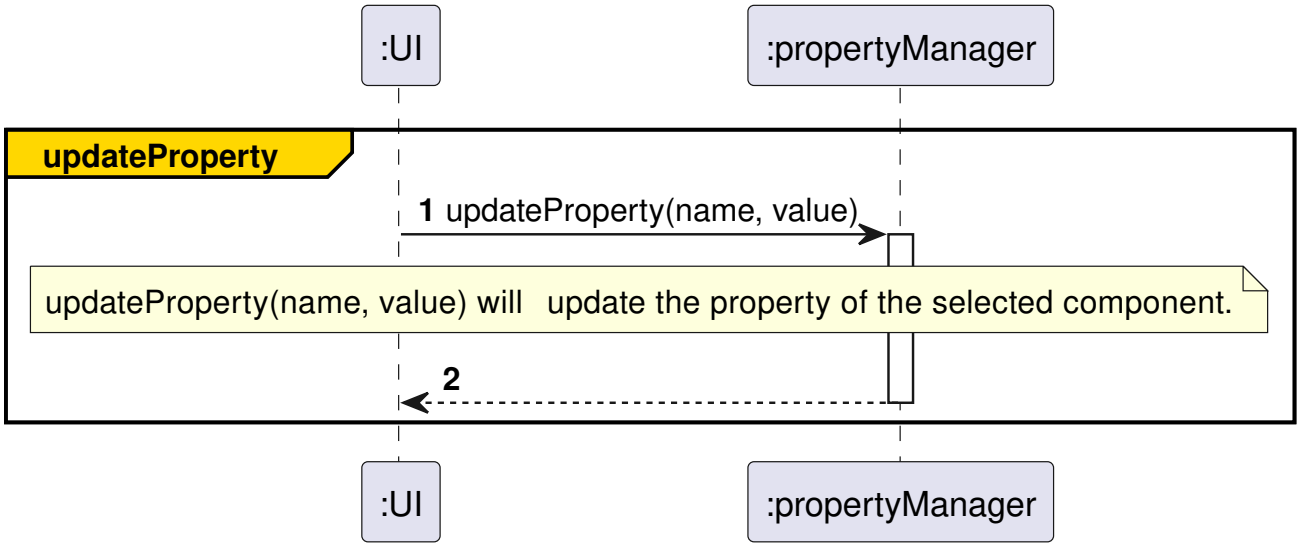


Figure 22: updateProperty

12 Design Class Model

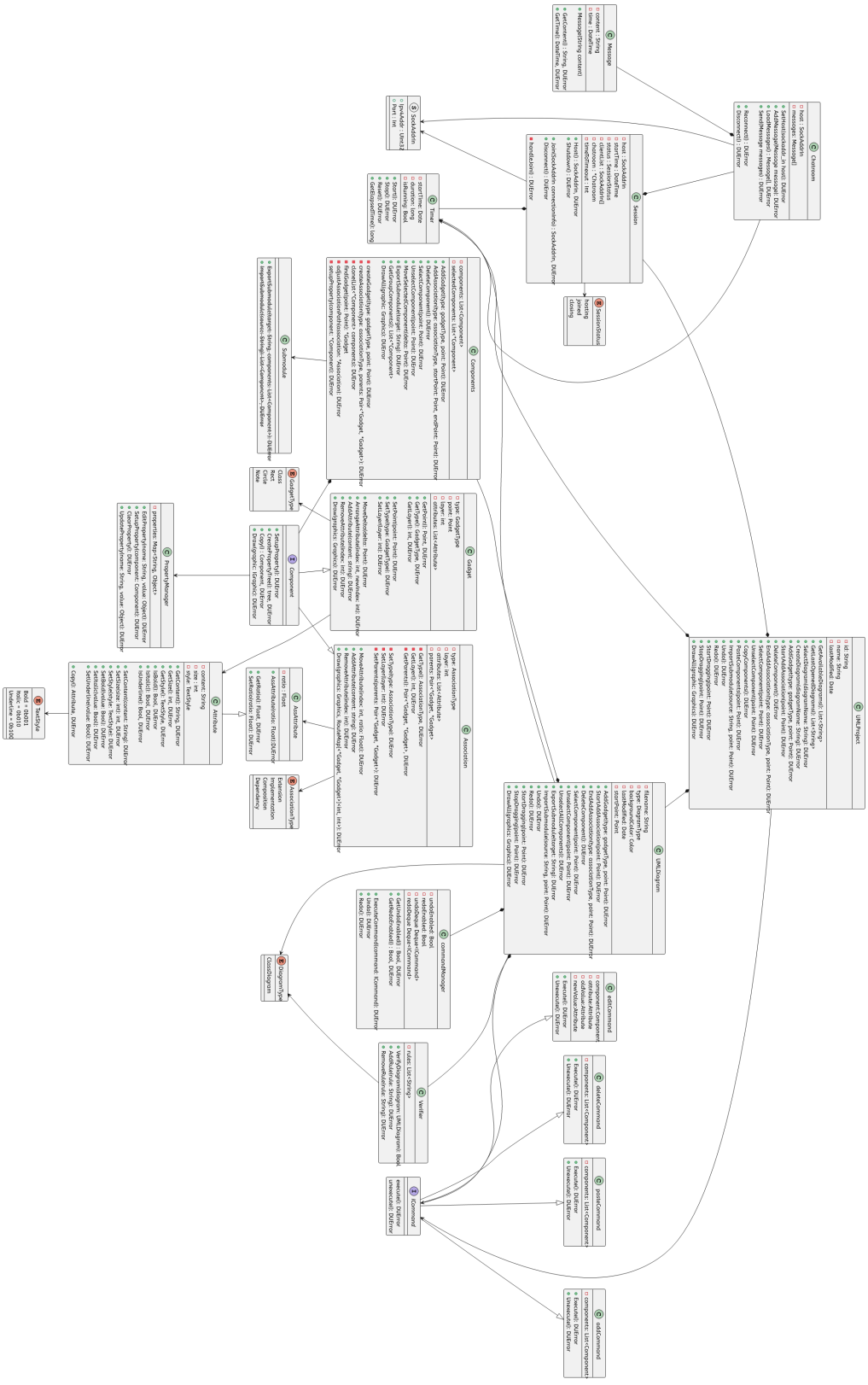


Figure 23: Design Class Diagram

13 Implementation Class Model

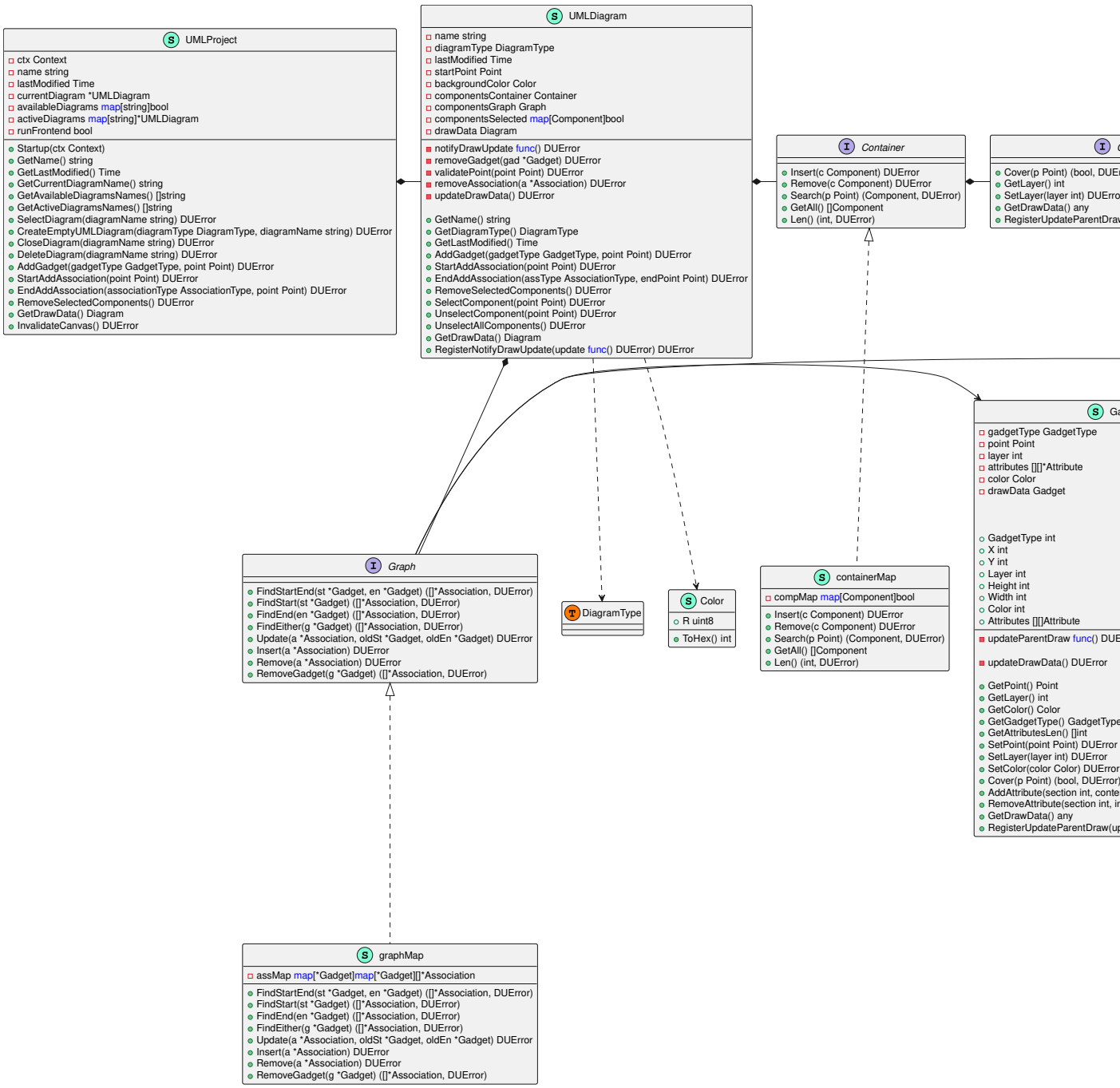


Figure 24: Design Class Diagram

HW5 TODO

14 Misc

14.1 View online

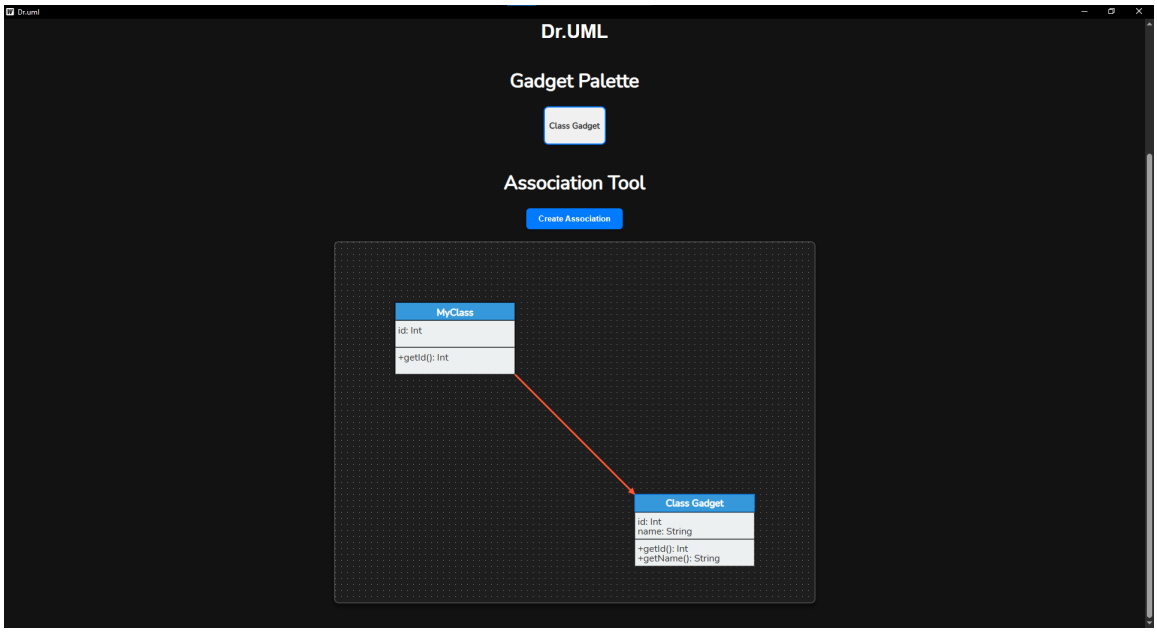
Since the report contains many images, we suggest visiting the GitHub repository to view higher-resolution versions.

15 Calculate Line of Code

No	Class Name	Number of methods	Line of Code in Class
1	AssAttribute	10	98
2	Attribute	18	160
3	Association	17	185
4	Component	5	17
6	Gadget	9	133
7	AssociationGraph	8	18
8	AssociationMap	10	178
9	Components	10	154
10	ComponentsContainer	5	15
11	ContainerMap	6	70
12	UMLDiagram	10	123
13	UMLProject	12	140
14	ConnectionError	2	13
15	DUError	1	6
16	FileIOError	3	13
17	InvalidArgumentError	3	13
18	MemoryFullError	3	13
19	FileIOError	3	13
20	SendError	3	13
21	Point	5	33
Total	-	143	1408

16 Programming

16.1 Snapshots of System Execution



16.2 Source Code Listing

```go
package component

import (
    "Dr.uml/backend/component/attribute"
    "Dr.uml/backend/drawdata"
    "Dr.uml/backend/utils"
    "Dr.uml/backend/utils/duerror"
)

type GadgetType int

const (
    Class               GadgetType = 1 << iota // 0x01
    supportedGadgetType            = Class
)

type Gadget struct {
    gadgetType       GadgetType
    point            utils.Point
    layer            int
    attributes       [][]attribute.Attribute // Gadget have multiple sections, each section have multiple attr
    color            utils.Color
    drawData         drawdata.Gadget
    updateParentDraw func() duerror.DUError
}

/*
component interface
*/

func (g *Gadget) Cover(p utils.Point) (bool, duerror.DUError) {
    tl := g.point                                                                    // top-left
    br := utils.AddPoints(g.point, utils.Point{X: g.drawData.Width, Y: g.drawData.Height}) // bottom-right
    return p.X >= tl.X && p.X <= br.X && p.Y >= tl.Y && p.Y <= br.Y, nil
}

func (g *Gadget) GetLayer() (int, duerror.DUError) {
    return g.layer, nil
}
```

```go
package component

import (
    "Dr.uml/backend/component/attribute"
    "Dr.uml/backend/drawdata"
    "Dr.uml/backend/utils"
    "Dr.uml/backend/utils/duerror"
)

type AssociationType int

const (
    Extension               = 1 << iota // 0x01
    Implementation          = 1 << iota // 0x02
    Composition             = 1 << iota // 0x04
    Dependency              = 1 << iota // 0x08
    supportedAssociationType = Extension | Implementation | Composition | Dependency
)

type Association struct {
    assType          AssociationType
    layer            int
    attributes       []*attribute.AssAttribute
    parents          [2]*Gadget
    drawdata         drawdata.Association
    updateParentDraw func() duerror.DUError
}

// Constructor
func NewAssociation(parents [2]*Gadget, assType AssociationType) (*Association, duerror.DUError) {
    if assType&supportedAssociationType != assType || assType == 0 {
        return nil, duerror.NewInvalidArgumentError("unsupported association type")
    }
    if parents[0] == nil || parents[1] == nil {
        return nil, duerror.NewInvalidArgumentError("parents are nil")
    }
    a := &Association{
        parents: [2]*Gadget{parents[0], parents[1]},
    }
    a.updateDrawData()
    return a, nil
}

// Getters
func (this *Association) GetAssType() AssociationType {
    return this.assType
}

```

17 Unit Testing

17.1 Snapshot

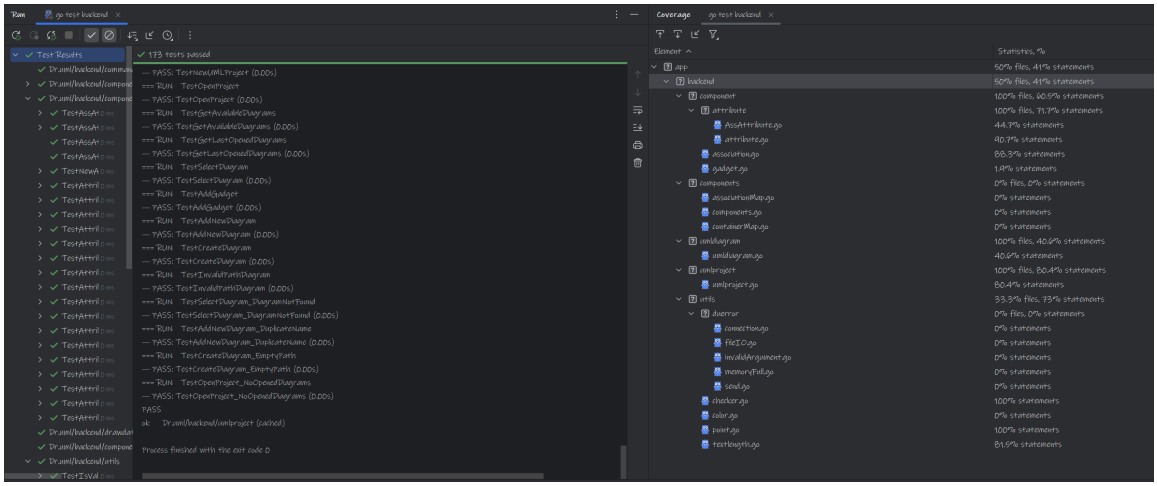


Figure 25: Unit test snapshot

17.2 Code listing

```
run test | debug test
func TestAttribute_GetContent(t *testing.T) {
    tests := []struct {
        name      string
        attribute Attribute
        expected  string
        hasError  bool
    }{
        {
            name:      "Valid content",
            attribute: Attribute{content: "testContent"},
            expected:  "testContent",
            hasError:  false,
        },
        {
            name:      "empty content",
            attribute: Attribute{content: ""},
            expected:  "",
            hasError:  false,
        },
    }

    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            result, err := tt.attribute.GetContent()
            if (err != nil) != tt.hasError {
                t.Errorf("unexpected error: %v", err)
            }
            if result != tt.expected {
                t.Errorf("expected %v, got %v", tt.expected, result)
            }
        })
    }
}
```

Figure 26: Unit test code

for Chinese

Info	Total
production code	1648
class in production code	15
methods in production code	183
number of unit tests	173
lines of unit tests	2036
蕭耕宏's time effort	48
張庭瑋's time effort	48
黃冠鈞's time effort	48
吳宥駒's time effort	48
total time effort	192

18 Misc

18.1 View online

Since the report contains many images, we suggest visiting the GitHub repository to view higher-resolution versions.

Deadline-Driven Development

References