Interactr: an interaction diagram drawing tool

Software-ontwerp
2017-2018

Iteration 3

Contents

1 Introduction

For the course *Software-ontwerp*, you will design and develop *Interactr*, a UML interaction diagram drawing tool. The main challenge will be the user interface layer, which you will design from scratch. In Section 2, we explain how the project is organized, discuss the quality requirements for the software you will design and develop, and describe how we evaluate the solutions. In Section 3, we show a diagram of the domain model and explain the problem domain of the application. The use cases are described in detail in Section 4. Finally, we specify some implementation constraints in Section 5.

2 General Information

In this section, we explain how the project is organized, what is expected of the software you will develop and the deliverables you will hand in.

2.1 Team Work

For this project, you will work in groups of four. Each group is assigned an advisor from the educational staff. If you have any questions regarding the project, you can contact your advisor and schedule a meeting. When you come to the meeting, you are expected to prepare specific questions and have sufficient design documentation available. *If the design documentation is not of sufficient quality, the corresponding question will not be answered.* It is your own responsibility to organize meetings with your advisor and we advise to do this regularly. Experience from previous years shows that groups that regularly meet with their advisors produce a higher quality design. If there are problems within the group, you should immediately notify your advisor. Do not wait until right before the deadline or the exam!

To ensure that every team member practices all topics of the course, a number of roles are assigned by the team itself to the different members at the start of each iteration (or shortly thereafter in case of the first iteration). A team member that is assigned a certain role will give the presentation or demo corresponding to that role at the end of the iteration. That team member is *not supposed to do all of the work concerning his task*! He must, however, take a coordinating role in that activity (dividing the work, sending reminders about tasks to be done, make sure everything comes together, etc.), and be able to answer most questions on that topic during the evaluation. The following roles will be assigned round-robin:

Design Coordinator The design coordinator coordinates making the design of your software.

Testing Coordinator The testing coordinator coordinates the planning, designing, and writing of the tests for the software.

Domain Coordinator The domain coordinator coordinates the maintenance of the domain model.

As already mentioned, the goal of these roles is to make every team member participate in all aspects of the development of your system. **During each presentation or demo, every team member must be able to explain the used domain model, the design of the system, and the functioning of your test suite.**

2.2 Iterations

The project is divided into 3 iterations. In the first iteration, you will implement the base functionality of the software. In subsequent iterations, new functionality will be added and/or existing functionality will be changed.

2.3 The Software

The focus of this course is on the *quality* (maintainability, extensibility, stability, readability,...) of the software you write. We expect you to use the development process and the techniques that are taught in this course. One of the most important concepts are the General Responsibility Assignment Software Principles (GRASP). These allow you to talk and reason about an object oriented design. **You should be able to explain all your design decisions in terms of GRASP**.

You are required to provide class and method documentation as taught in previous courses (e.g. the OGP course). When designing and implementing your system, you should use a *defensive programming style*. This means that the *client* of the public interface of a class cannot bring the objects of that class, or objects of connected classes, into an inconsistent state.

Unless explicitly stated in the assignment, you do not have to take into account persistent storage, security, multi-threading, and networking. If you have doubts about other non-functional concerns, please ask your advisor.

2.4 Testing

All functionality of the software should be tested. **For each use case, there should be a dedicated scenario test class.** For each use case flow, there should be at least one test method that tests the flow. Make sure you group your test code per step in the use case flow, indicating the step in comments (e.g. `// Step 4b`). Scenario tests should not only cover success scenarios, but also negative scenarios, i.e., whether illegal input is handled defensively and exceptions are thrown as documented. You determine to which extent you use unit testing. The testing coordinator briefly motivates the choice during the evaluation of the iteration.

Tests should have good coverage, i.e. a testing strategy that leaves large portions of a software system untested is of low value. Several tools exist to give a rough estimate of how much code is tested. One such tool is Eclemma[1]. If this tool reports that only 60% of your code is covered by tests, this indicates there may be a serious problem with (the execution of) your testing strategy. However, be careful when drawing conclusions from both reported high coverage

[1]http://www.eclemma.org

and reported low coverage (and understand why you should be careful). The testing coordinator is expected to use a coverage tool and briefly report the results during the evaluation of the iteration.

2.5 UML Tools

There are many tools available to create UML diagrams depicting your design. You are free to use any of these as long as it produces correct UML. One of these UML tools is Visual Paradigm. Instructions to run Visual Paradigm in the computer labs is described in the following file:

```
/localhost/packages/visual_paradigm/README.CS.KULEUVEN.BE
```

This file also contains the location of the license key that you can use on your own computer.

2.6 What You Should Hand In

Exactly *one* person of the team hands in a ZIP-archive via Toledo. The archive contains the items below and follows the structure defined below. **Make sure that you use the prescribed directory names.**

- directory `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `doc`: a folder containing the Javadoc documentation of your entire system
  - `diagrams`: a folder containing UML diagrams that describe your system (at least one structural overview of your entire design, and sufficient detailed structural and behavioural diagrams to illustrate every use case)
  - `src`: a folder containing your source code
  - `system.jar`: an executable JAR file of your system

When including your source code into the archive, make sure to *not include files from your version control system.* If you use subversion, you can do this with the the `svn export` command, which omits unnecessary repository folders from the source tree. Make sure you choose relevant file names for your analysis and design diagrams (e.g. `SSDsomeOperation.png`). You do **not** have to include the project file of your UML tool, only the exported diagrams. We should be able to start your system by executing the JAR file with the following command: `java -jar system.jar`.

Needless to say, the general rule that anything submitted by a student or group of students must have been authored exclusively by that student or group of students, and that accepting help from third parties constitutes exam fraud, applies here.

2.6.1 Late Submission Policy

If the zip file is submitted N minutes late, with $0 \leq N \leq 240$, the score for all team members is scaled by $(240 - N)/240$ for that iteration. For example, if your solution is submitted 30 minutes late, the score is scaled by 87.5%. So the maximum score for an iteration for which you can earn 4 points is reduced to 3.5. If the zip file is submitted more than 4 hours late, the score for all team members is 0 for that iteration.

2.6.2 When Toledo Fails

If the Toledo website is down – and *only* if Toledo is down – at the time of the deadline, submit your solution by e-mailing the ZIP-archive to your advisor. The timestamp of the departmental e-mail server counts as your submission time.

2.7 Evaluation

After iteration 1, and again after iteration 2, there will be an intermediate evaluation of your solution. An intermediate evaluation lasts 15 minutes and consists of: a presentation about the design and the testing approach, accompanied by a demo of the tests.

The intermediate evaluation of an iteration will cover only the part of the software that was developed during that iteration. Before the final exam, the *entire* project will be evaluated. It is your own responsibility to process the feedback, and discuss the results with your advisor.

The evaluation of an iteration is planned in the week after that iteration. Immediately after the evaluation is done, you mail the PDF file of your presentation to Prof. Bart Jacobs & Tom Holvoet and to your advisor.

2.7.1 Presentation Of The Current Iteration

The main part of the presentation should cover the design. The motivation of your design decisions *must* be explained in terms of GRASP principles. Use the appropriate design diagrams to illustrate how the most important parts of your software work. Your presentation should cover the following elements. Note that these are not necessarily all separate sections in the presentation.

1. An updated version of the domain model that includes the added concepts and associations.
2. A discussion of the high level design of the software (use GRASP patterns). Give a rationale for all the important design decisions your team has made.
3. A more detailed discussion of the parts that you think are the most interesting in terms of design (use GRASP patterns). Again we expect a rationale here for the important design decisions.
4. A discussion of the extensibility of the system. Briefly discuss how your system can deal with a number of change scenarios (e.g. extra constraints, additional domain concepts,...).
5. A discussion of the testing approach used in the current iteration.

6. An overview of the project management. Give an approximation of how many hours each team member worked. Use the following categories: group work, individual work, and study (excluding the classes and exercise sessions). In addition, insert a slide that describes the roles of the team members of the current iteration, and the roles for the next iteration. Note that these slides do not have to be presented, but we need the information.

Your presentation should not consist of slides filled with text, but of slides with clear design diagrams and keywords or a few short sentences. The goal of giving a presentation is to communicate a message, not to write a novel. All design diagrams should be *clearly readable* and use the correct UML notation. It is therefore typically a bad idea to create a single class diagram with all information. Instead, you can for example use an overview class diagram with only the most important classes, and use more detailed class diagrams to document specific parts of the system. Similarly, use appropriate interaction diagrams to illustrate the working of the most important (or complex) parts of the system.

2.8 Peer/Self-assessment

In order for you to critically reflect upon the contribution of each team member, you are asked to perform a peer/self-assessment within your team. For each team member (including yourself) and for each of the criteria below, you must give a score on the following scale: *poor/lacking/adequate/good/excellent*. The criteria to be used are:

- Design skills (use of GRASP and DESIGN patterns, ...)
- Coding skills (correctness, defensive programming, documentation,...)
- Testing skills (approach, test suite, coverage, ...)
- Collaboration (teamwork, communication, commitment)

In addition to the scores themselves, we expect you to briefly explain for each of the criteria why you have given these particular scores to each of the team members. The total length of your evaluation should not exceed 1 page.

Please be fair and to the point. Your team members will not have access to your evaluation report. If the reports reveal significant problems, the project advisor may discuss these issues with you and/or your team. Please note that your score for this course will be based on the quality of the work that has been delivered, and not on how you are rated by your other team members.

Submit your peer/self-assessment by e-mail to both Prof. Bart Jacobs & Tom Holvoet and your project advisor, using the following subject: **[SWOP] peer-/self-assessment of group $groupnumber$ by $firstname$ $lastname$**.

2.9 Deadlines

- The deadline for handing in the ZIP-archive on Toledo is **25 May, 2018, 6pm**.

- The deadline for submitting your peer/self-assessment is **27 May, 2018, 6pm**, by e-mail to both your project advisor and Prof. Bart Jacobs & Tom Holvoet.

3 Interactr

The goal of the project is to develop a UML interaction diagram drawing tool. A UML interaction diagram shows a number of *parties* and a sequence of *messages* sent between these parties.

- A party is either an *actor* or an *object*. An actor is shown as a stick figure; an object is shown as a rectangle. Each party has a *label*, consisting of an optional *instance name*, followed by a colon, followed by a *class name*. An instance name starts with a lowercase letter; a class name starts with an uppercase letter. An actor's label is shown below the stick figure; an object's label is shown inside the rectangle.

- A message has a sender party, a receiver party, and a label. A message is either an invocation message (continuous line) or a result message (dashed line). The label of an invocation message consists of a method name and an argument list. A method name starts with a lowercase letter and consists only of letters, digits, and underscores. An argument list is a parenthesized, comma-separated list of arguments. An argument is any sequence of characters, not including commas or parentheses. The label of a result message is unconstrained.

We call the first message's sender the *initiator* of the interaction. We associate a *call stack* (a sequence of parties) with each point during the interaction as follows:

- The initial call stack contains just the initiator of the interaction.

- The call stack after an invocation message equals the call stack before the message, with the message's receiver appended to the end.

- The call stack after a result message equals the call stack before the message, with the last element removed.

The sequence of messages must be *single-threaded*: a message's sender must be at the top of the call stack before the message, and a result message's receiver must be at the top of the call stack after the message. Furthermore, the sequence must be *complete*: the final call stack must be equal to the initial call stack.

Note: any feature of UML interaction diagrams not mentioned here need not be supported by your system in this iteration. In particular, your system need not support conditional messages or iterated messages in this iteration. Also, in this iteration messages from a party to itself need not be supported.

Your system must allow the user to view and edit an interaction both in the form of a *sequence diagram* and in the form of a *communication diagram*.

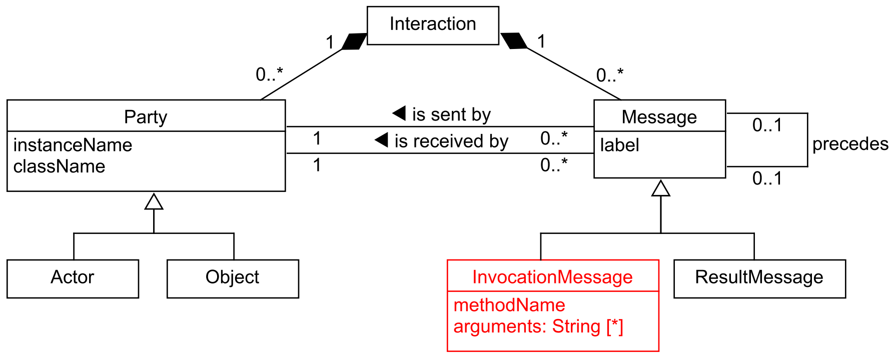

Figure 1: Domain model

- In a sequence diagram, the parties are shown side by side at the top of the diagram. A *lifeline* for each party is shown below the party. Each message is shown in a separate row, as an arrow from the sender party's lifeline to the receiver party's lifeline, with the label shown above the arrow. On each lifeline, between each incoming invocation message and the corresponding outgoing result message, an *activation bar* is shown. The arrows for messages incoming to or outgoing from this activation depart from the edges of this activation bar. If no activation is active when an incoming invocation message arrives, the activation bar is centered around the party's lifeline. If activations are active, the activation bar is centered around the innermost (i.e. most recently activated) active activation's right-hand edge. All activation bars have the same width.

- In a communication diagram, the parties are shown at user-specified locations on the diagram. If messages flow between two parties, a link (a continuous line) is drawn that connects them, and for each invocation message an arrow indicating the direction of the message, and the message label are shown near the link. Result messages are not shown in a communication diagram. Each message's label is preceded by its *message number*: the N'th invocation message sent by the initiator while no invocations were active has message number N; the N'th invocation message sent by an activation activated by an invocation message with message number η has message number $\eta.N$.

Figure 1 shows the domain model of Interactr.

Clearly, at this level of abstraction, the problem domain is quite simple. The main challenge of this assignment, then, lies not in the design of the domain layer of your system, but in the design of the user interface layer.

Your system shall allow the user to edit multiple interactions, and multiple diagrams of the same interaction, simultaneously. Each diagram is shown in a separate subwindow of the application window. Each subwindow can be moved (by dragging the titlebar), resized (by dragging an edge or a corner), and closed (by clicking the Close button next to the titlebar). Subwindows can overlap.

Subwindows are drawn in a particular order, against a gray background; activating a subwindow (by clicking any part of it) brings it to the front. In this iteration, you do not need to support scrolling (either the application window or the subwindows).

At system startup, there are no open interactions, i.e., no subwindows; only the gray background is visible. The gray background cannot be interacted with. The only possible interaction at this point is to enter the Ctrl+N key combination, which shows a new subwindow which shows a sequence diagram of a new, empty interaction. The diagram type of the active subwindow can be changed, as before, by pressing the Tab key. Pressing Ctrl+D while a subwindow is active shows a new subwindow which shows another diagram of the same interaction shown by the active subwindow. The new diagram is initially identical to the existing one, but after creation the diagram type and the positions of the elements can be mutated independently of the existing diagram.

All subwindows that show diagrams of the same interaction remain in sync at all times: they show the same parties, in the same form (actor or object), with the same labels, and the same sequence of messages, with the same labels, between them. The diagram type and the positions of the elements can, however, be mutated independently in each subwindow.

However, if in some subwindow a label is being edited and an edit operation (i.e., a character insertion or deletion) causes the value to become invalid, the invalid value is shown in red and is not propagated to the other subwindows. Furthermore, the subwindow is locked into label editing mode until the value is again valid, but interaction with other subwindows is not constrained. If a later edit operation in the same subwindow causes the value to become valid again, the valid value is propagated to the other subwindows that show the same interaction at this point. Also, when a valid label value is propagated to a subwindow, the label value shown by that subwindow is replaced by the propagated value, regardless of whether the subwindow was showing a valid or an invalid value.

Some characteristics of diagrams, parties, and messages can be edited directly in the diagrams as well as through *dialog boxes*. A dialog box is another kind of subwindow: it has a title bar and a close button, it can be moved and resized, and clicking it moves it to the top of the stack of subwindows. However, instead of showing a diagram, a dialog box shows a number of *controls* through which certain characteristics of the corresponding diagram or diagram element can be edited. Any edits performed on a characteristic through a dialog box control are visible immediately in all subwindows (diagram subwindows and dialog boxes) that show that characteristic, except if the edits result in an invalid value. In that case, any edits performed through other subwindows that result in a valid value cause the invalid value to be overridden.

A dialog box for a diagram shows two *radio buttons* for choosing between a sequence diagram and a communication diagram. A dialog box for a party shows two *text boxes* and two radio buttons, for editing the instance name and the class name and for choosing between the actor and object form, respectively. A dialog box for an invocation message shows a text box for the method name, a *list box* for the arguments, a text box and a *button* for adding a new argument to the end of the argument list, buttons for moving the selected argument up or down in the argument list, and a button for deleting the selected argument. The latter three buttons are *disabled*, indicated by being grayed out, if no argument is currently selected.[2] A dialog box for a result message shows a text box for the label.

The user can navigate between the controls of a dialog box using the Tab key. They can activate a button and select a radio button using the space bar, and select a list box element using the arrow keys. These operations can also be performed using mouse clicks.

A new dialog box for the currently selected diagram element, or for the diagram shown by the currently selected diagram subwindow if no element within that subwindow is selected, is opened by pressing Ctrl+Enter. Many dialog boxes may be open for the same diagram or diagram element at any given time.

4 Use Cases

Figure 2 shows the use case diagram for Interactr. The following sections describe the various use cases in detail.

Notes:

- A system's requirements can be specified at various levels of abstraction with respect to the nature of the interface between the system and its environment (e.g. a human user). Often the specification abstracts over the nature of the interface to focus on the aspects that are most relevant to the usefulness of the system. However, in this assignment, since the main challenge concerns the design of the code that implements the user interface, we specify the interaction with the user in unusually specific detail.

- While the tool you develop should be functional, the user interface need not be of the level of "finish" that would be expected of a commercial product. For example:

  - The text cursor need not blink.
  - You need not support scrolling the diagram view if the diagram is larger than the canvas (after the user resizes the window).
  - You need not support labels that are more than 30 characters long.
  - You need not (in this iteration) provide a menu, Save/Open functionality, printing functionality, etc.

4.1 Use Case: Switch Diagram Type

Precondition: A subwindow is active.

Main Success Scenario:

1. The user presses the Tab key.
2. If the subwindow was showing a sequence diagram, it is converted into a communication diagram showing the same interaction, and vice versa.

[2] Your system need not support editing an argument. Instead, the user can delete the argument and add a new one with the updated text.

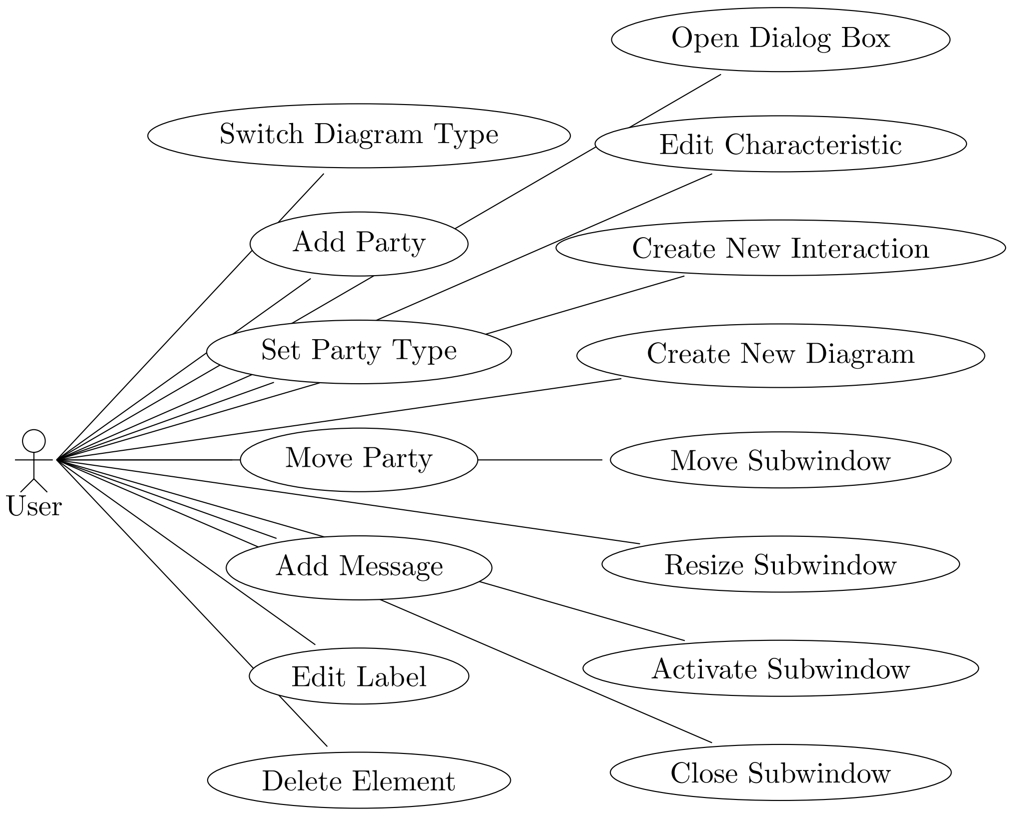

Figure 2: Use case diagram for Interactr.

4.2 Use Case: Add Party

Main Success Scenario:

1. The user double-clicks in an empty area of the parties row of the sequence diagram, or in an empty area of the interaction diagram.
2. The system adds a new party to the diagram, in the form of an object, with an empty label, at the location of the double-click event. The system shows a text cursor where the label belongs to indicate that the user can now type the party's label.
3. The user enters a label for the party.
4. The system shows the updated label as it is being entered. The subwindow refuses to exit label editing mode (i.e. it accepts only character and Backspace key presses, and no other keyboard or mouse events) while the label is invalid (i.e. the instance name, if present, does not start with a lowercase letter, or no colon is present, or the class name does not start with an uppercase letter.

4.3 Use Case: Set Party Type

Main Success Scenario:

1. The user double-clicks a party (not its label).
2. If the party was shown as an object, the subwindow now shows it as an actor, and vice versa.

4.4 Use Case: Move Party

Main Success Scenario:

1. The user moves the mouse cursor over a party (not its label), presses the left mouse button, and then moves the mouse cursor to an empty area of the parties row of the sequence diagram, or an empty area of the interaction diagram, while keeping the mouse button pressed.
2. The system shows the party's new position while the user is moving the mouse cursor.
3. The user releases the mouse button.
4. The system updates the party's position.

4.5 Use Case: Add Message

Main Success Scenario:

1. The user presses the left mouse button while the mouse cursor is (approximately) over some party's lifeline in a sequence diagram, not inside the row of some existing message (but potentially between the rows of two existing messages), and then moves the mouse to (approximately)

the corresponding position on another party's lifeline before releasing the mouse button.[3]

2. The system inserts an invocation message from the first party to the second party into the sequence of messages, with an empty label. It also inserts the corresponding result message from the second party to the first party, immediately below it. It shows the updated diagram. The system shows a text cursor where the invocation message's label belongs, to indicate that the user can now type the message's label.
3. The user enters a label for the party.
4. The system shows the updated label as it is being entered.

Extensions:

2a. The first party is not at the top of the call stack at the indicated position in the interaction.

   1. The mouse gesture has no effect. The system shows the diagram, unchanged.

4.6 Use Case: Edit Label

Main Success Scenario:

1. The user clicks the label of a party or a message, or the location where the label belongs if the label is empty.
2. The system indicates graphically that the clicked element is now the selected element of the diagram.
3. The user clicks the same label again.
4. The system shows a text cursor at the end of the label, to indicate that the user can now edit the label.
5. The user enters characters to add them to the label, and/or the Backspace key to remove the last character from the label.
6. The system shows the updated label as it is being edited. The subwindow refuses to exit label editing mode (i.e. it accepts only character and Backspace key presses, and no other keyboard or mouse events) while the label is invalid. Whenever the label value is valid, the system immediately shows the updated value in all subwindows that show the interaction being edited.

4.7 Use Case: Delete Element

Main Success Scenario:

1. The user clicks the label of a party or an invocation message.
2. The system indicates graphically that the clicked element is now the selected element of the diagram.

[3]Your system need not support adding messages to a communication diagram, except by adding messages to a sequence diagram in another subwindow showing the same interaction.

3. The user presses the Delete key.
4. The system removes the element from the diagram, as well as all other elements that directly or indirectly depend on it: a message depends directly on the party that sends it, the party that receives it, and the invocation message that initiates the activation that sends it.

4.8 Use Case: Create New Interaction

Main Success Scenario:

1. The user enters the Ctrl+N key combination.
2. The system shows a new subwindow on top of the existing subwindows, showing a sequence diagram for a new, empty interaction. The new subwindow is now the active subwindow.

4.9 Use Case: Create New Diagram

Precondition: A subwindow is active.

Main Success Scenario:

1. The user enters the Ctrl+D key combination.
2. The system shows a new subwindow on top of the existing subwindows, showing an identical copy of the diagram shown by the previously active subwindow, showing the same interaction. The new subwindow is now the active subwindow.

4.10 Use Case: Move Subwindow

Main Success Scenario:

1. The user moves the mouse cursor over the titlebar of a subwindow, depresses the mouse button, moves the mouse cursor, and then releases the mouse button.
2. The system activates the subwindow and moves it by the same displacement as the displacement performed by the mouse cursor while the mouse button was depressed.

4.11 Use Case: Resize Subwindow

Main Success Scenario:

1. The user moves the mouse cursor over an edge or a corner of a subwindow, depresses the mouse button, moves the mouse cursor, and then releases the mouse button.
2. The system updates the position of the edge or corner of the subwindow accordingly, while leaving the positions of the other edges, or of the opposite corner, respectively, unchanged.

4.12 Use Case: Activate Subwindow

Main Success Scenario:

1. The user moves the mouse cursor over any part of a subwindow and depresses a mouse key.
2. The system sets the subwindow as the the active subwindow and shows it in front of all other subwindows, in addition to performing any reactions to the mouse gesture described in other use cases.

4.13 Use Case: Close Subwindow

Main Success Scenario:

1. The user clicks a subwindow's Close button.
2. The system no longer shows this subwindow.

4.14 Use Case: Open Dialog Box

Precondition: A party, message, or diagram has been selected.

Main Success Scenario:

1. The user enters the Ctrl+Enter key combination.
2. The system shows a new dialog box on top of the existing subwindows, showing a set of controls for editing certain characteristics of the selected diagram or diagram element.

4.15 Use Case: Edit Characteristic

Precondition: A dialog box is visible.

Main Success Scenario:

1. The user selects a radio button, activates a button, selects a list box element, or edits the text in a text box.
2. The system shows the resulting changes in the characteristics in all subwindows that show the characteristic, except if the edits result in an invalid value.

5 Implementation

You must implement your system in a statically typed, object-oriented programming language.

Since the main intended challenge of this assignment is that you design your user interface layer from scratch, for this assignment you are not allowed to use an existing GUI toolkit, such as Java's Swing/AWT or SWT, or the .NET Framework's Windows Forms or Windows Presentation Framework. Instead, we require that you use only the CanvasWindow Java class that we provide, or an equivalent

API that you implement yourself on another platform, to implement the user interface. (When using the provided CanvasWindow Java class, you can also use the AWT elements that are necessary to use this class, but you cannot use the AWT or Swing component hierarchies.)   Good luck!

The SWOP Team members