# Report Final Assignment

Student information

*Randy Pöttgens*

*851941098*

*Ivo Willemsen*

*851926289*

***Scenario 3, Interactive Navigation***

Approach

<*Geef aan hoe jullie de opdracht hebben aangepakt en wie wat heeft gedaan, maximaal 1 A-4. Geef expliciet aandacht aan de volgorde van activiteiten*>

|  |  |
| --- | --- |
|  | Assignment 1: Problem analysis Jabberpoint is a simple slide show application that can read a slide show from a source allows the user to navigate through the slides and can save the state of the running slide show to the source again.  This problem analysis is split into two parts: The first part focuses on the identification of the concepts, the entities. The latter part will elaborate on the rules that can be extracted from the case description.  Assumptions are made when necessary. Concepts |

|  |  |
| --- | --- |
| *slide show*  *head, title, theme*  *slide*  *thread*  *slide item*  *text item*  *level*  *bitmap item*  *action item*  *displayable item*  *style of a displayable item* | The main concept is the **slide show**. A slide show is a presentation of a series of slides (still images) on the screen, in a *prearranged sequence*. A slide show consists of the following parts:   * A **head**, which consists of a **title** and a **possible theme** (ask for clarification) * A list of **slides**. There must be *at least one slide* present (ask for clarification) in the slide show. Slides in a slide show have a prearranged order (first slide will have sequence no. 1 and the last slide sequence no. n) * A list of **threads** (ask for clarification).   A slide contains a number of **slide items**, which are items that are displayed on the slide. Slide items are displayed one after the other in a predefined order. The user will not have control over when or how the slide items are displayed.  A slide item can have three forms:   * A **text item**. An item that consists of a simple text (string) and has a certain **level** * A **bitmap item**. An item that represents an image. Also a bitmap item has a certain level * An **action item**. An action item cannot be displayed, but contains another slide item, which can be a text, bitmap or action item. The “leaf” item of an action item is a non-action item, i.e. a text item or a bitmap item. Looking at it in a different way: A text item or a bitmap item can have (optionally) 1 or more actions attached to it. It must be noted that an action can only have one child action. An action item doesn’t have a level, as it is not associated with the displaying of an item on the screen   From the above, it can be deduced, that text and bitmap items are **displayable items**, and that action items are not directly displayed on the screen, but contain a displayable item at the leaf-level. The action item add a border to the displayable item to indicate that one or more actions will be performed when the user click on the item.  A displayable item has a level and an action item does not have a level. A level is associated with a certain **style**. So displayable items have associated styles. An action item is not associated with a style directly.  A text item is styled in a different way than a bitmap item. A style for a text item can for example have a certain color, while a color for a bitmap style is not appropriate, as the coloring aspect of a bitmap is inherently determined by the bitmap itself. The following table shows the characteristics of both type of styles: |

|  |  |
| --- | --- |
| **Type of style** | **Characteristics** |
| Common style | * X-padding (“indent”). Padding on the x-axis, amount of space that is taken into account from the beginning of the containing frame * Y-padding (“leading”). Padding on the y-axis, amount of space that is taken into account from the y-value + height of the previous item |
| Text style | * Common style characteristics (see above) * Font size * Font color |
| Bitmap style | * Common style characteristics (see above) |

*Table 1: Styles types*

The following constrains and additional functionalities are valid:

* X- and y-values are deduced, based on the containing frame, the level associated with the style, and the sequence number of the item
* When drawing items, the scale of the screen is also taken into account.
* Styles will be hard-coded in the application

The next figure can be used to put these characteristics in perspective.

Text item

Bitmap item

leading

indent

scale

font color

font size

*Figure 1: Style characteristics*

|  |  |
| --- | --- |
| *action*  *navigation action*  *current slide*  *absolute, relative navigation action*  *slide show persistence action*  *source*  *auxiliary action* | An important aspect of this assignment is the concept of “action”. The first type of **action** is the **navigation** action. The result of this action is a change of the **current** slide. The current slide in a presentation is the slide that is being displayed at a certain moment in time. The current slide is a feature that should be maintained throughout different presentation sessions and as such (it is assumed), should be saved upon user request (By using the File | Save menu item). When a slide show is retrieved from the source, the current slide is determined and the navigation action to go to the indicated slide is performed.  The following navigation actions should be supported by the application:   * Go to next slide * Go to previous slide * Go to first slide * Go to last slide * Go to slide i   Navigation actions can either be **absolute** or **relative**. A relative navigation action takes the current slide into account. An absolute navigation action does not take the current slide into account, but indicates directly the slide that should be navigated to.  A second type of action is an action that operates on the level of slide shows persistence. A slide show can be **opened** or **saved**. On saving a slide show, the current slide is recorded in the **source**. Slide shows can be saved to or retrieved from different types of sources, like an **XML format** or a predefined **Demo format** (hard coded in the application). Of course, adding a different source to the application, like a database format, should require minimal effort and not affect the design of application in a major way. When saving to a Demo format, the application should remember the current slide during the duration of the user session. The application can be started with an argument that indicates the type of source that is being used. In case “demo” is used, the internal Demo format is used. When the argument “xml” is used, the user will be provided with a dialog box where he/she can select the file that contains the slide show.  Finally, the last type of action is an auxiliary action. An auxiliary action for example is a beep sound, or a graphical effect. |

### Rules

This paragraph focuses on the rules that must be enforced. These rules are extracted from the case description and, if not clear, assumed.

* A save action can only be issued by the user by selecting the option “File|Save” from the menu
* When a slide show is saved, the current slide number is stored in the source
* An open action can be issued by the user by selecting “File|Open” from the menu and by the system when the slide show is being read from the source. In this case, the open action must be issued when the system finds the action tag (xml) or otherwise (database, not defined yet). When an open action is issued, either the user must provide the name of the slide show in a dialog window, or the system must be provided with the name of the slide show in the action tag
* When a slide show is opened, the stored current slide number is read from the source and the system navigates to the indicated stored slide number
* An absolute navigation action is either a “go to first slide”, “go to last slide” or “go to slide i” navigation action. The first two don’t require extra parameters. The latter requires a user to provide the slide number in a dialog window or the system must provide the page number in the corresponding action tag
* A relative navigation action is either a “go to next slide” or a “go to previous slide” navigation action and both don’t require extra information
* Any kind of navigation action can be issued by the user by using the menu, keyboard or clicking on the text item or bitmap item that has an associated navigation action attached to it. It can also be issued by the system, when the application opens a slide show and the appropriate action tag indicates a certain navigation action
* When the application reads a slide show from a source, and an open action is encountered in an action tag, subsequent action tags are ignored, as these additional action tags don’t operate on the same slide show anymore
* When the application reads a slide show from a source with name x, and an embedded action in one of the slide items instructs to open the same slide show with name x, the system will raise an error and not load the slide show in the embedded action
* An auxiliary action can only be issued by the system (like a sound) when reading the slide show from the source. A user cannot issue such action directly (apart from opening a slide show)

The above rules are summarized in the following CVA table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Action** | **Type** | **Additional**  **action** | **Parameter** | **Actor** |
| go to first, last slide | absolute  navigation |  | page | * user (key, link, menu) * system (action in xml) |
| go to slide i | absolute  navigation | ask for page no. / get page no. from action in xml | page | * user (key, link, menu) * system (action in xml) |
| go to next, previous slide | relative navigation |  | current slide number | * user (key, link, menu) * system (action in xml) |
| open slide | Slide show persistence | ask for source selection | name of the source (file) | * user (menu) * system (action in xml) |
| save slide | Slide show persistence |  | name of the source (file) | * user (menu) |
| auxiliary action | auxiliary action |  |  | * system (action in xml) |

*Table 1: Action rules*

## Assignment 2: Design

The following activity diagram depicts the activities that can be identified on a very high level:



*Figure x: High-level Activity Diagram*

The MVC Pattern will be used to separate entities into Model, Viewer and Controller entities:



*Figure x: MVC Class diagram*

Slide/SideItem is responsible for initiating the draw methods in the DrawingDriver according to GRASP-guidelines according to the Expertpattern. Controller entities communicate with the ActionFactory. The responsibility of the ActionFactory is to generate the correct Action entities that know how to perform certain actions. An Action will invoke for example a method on the SlideShow to navigate to the next slide. The SlideShow delegates the responsibility to the Slide.

There is no need for any entities in the Model part of the domain model to be observed for changes. Therefore, Observer Pattern is not necessary in our opinion. One could argue that the invocation of a draw method on the SlideItem for example could be a reason to notify Observers (entities in the View part), but as far as it stands right now, decoupling the Model entities Slide and SlideItem from the View concrete implementation SwingDrawingDriver by using the interface DrawingDriver, is good enough.

The following domain model represents a first attempt to model entities and relationships in the Model part of the domain:



*Figure x: Items and actions, first alternative*

Actions are attached here to the slide item. A slide item has 0, 1 or more actions attached to it. In case a slide item has an action attached to it, a border should be drawn to indicate to the user that clicking on the item will results in 1 or more actions.

BitMapItem and TextItems have different kind of Styles. A TextItem has a TextItemStyle, which includes a font color and a font size, which a BitMapStyle is lacking.

In the above domain model, a decorator pattern is used to decorate the bitmap or text item with a border. Actions are attached to the ActionItemDecorator abstract class.

An alternative to this could be the figure that is depicted in the following figure, where no Decorator Pattern is used. In this situation, A SlideItem has 0, 1 or more Actions attached to it by means of a composition.



*Figure x: Items and actions, second alternative*

Actually both are valid alternatives, and one of the options must be chosen. The other alternative will be moved to the section regarding design decisions later on, including an explanation why we have chosen the alternative.

As it looks like right, we will go for the Decorator Pattern.

The following figure depicts the action hierarchy:



*Figure x: Action hierarchy*

The following figure shows how slide items and slides in the domain model are separated from the drawing implementations. The idea is that in the future perhaps other implementations are introduced, like JavaFx, In that case, only an extra JavaFXDrawingDriver must be created that encapsulates that specific logic.



*Figure x: Slide/SlideItem Bridge to DrawingDriver*

Slide and SlideItem are entities that belong to the Model part of the MVC Pattern. Model entities should not have direct navigable references to Controller and View entities. That is additional reason to introduce the DrawingDriver.

## Assignment 3: Design decisions

## Assignment 4: Source code

Good practice rules:

Parameters that are only used as input, should be made final to enforce this

Variables

Parameters that never should be null, should be tagged with the annotation @NotNull