# 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 slideshow application that can read a slideshow from a source allows the user to navigate through the slides and can save the state of the running slideshow 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 |

|  |  |
| --- | --- |
| *slideshow*  *title*  *theme*  *background color*  *slide*  *slide item*  *text item*  *level*  *bitmap item*  *action*  *style* | The main concept is the **slideshow**. A slideshow is a presentation of a series of slides (still images) on the screen, in a *prearranged sequence*. A slideshow consists of the following parts:   * A **title**. The title of the selected slideshow will be displayed in the frame of the application * A **theme**. A slide is configured with a certain theme. The theme determines the **background color** of the slideshow. All slides in a slideshow will have that same background color * A list of **slides**. There must be *at least one slide* present in the slideshow. Slides in a slideshow have a prearranged order (first slide will have sequence number 1 and the last slide sequence number n)   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, except for the fact that slide items can be assigned a certain level, which determines the way slide items look (style) and the position on the screen.  A slide item can have two 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   Slide items can have 0, 1 or more **actions** attached to them which can be activated by clicking on either the text or the bitmap. The actions are preconfigured and this configuration determines the order that the actions are performed upon activation. Actions are discussed in more details later.  As said, all items have an associated level, and this level determines the **style**. How a slide item is styled is solely dependent on the level of the slide item.  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*  *slideshow 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 slideshow 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 slideshow sessions and as such (it is assumed), should be saved upon user request (By using the File | Save menu item). When a slideshow 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 slideshows persistence. A slideshow can be **opened** or **saved**. On saving a slideshow, the current slide is recorded in the **source**. Slideshows 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. Saving to a demo format will not be possible, as it does not add any value to the application.  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.

Persistence actions rules

The following rules impact the way how slideshows are read from and written to sources (demo format, xml format or database format):

* A “slideshow save” action can only be issued by the user by selecting the option “File|Save” from the menu
* When a slideshow is saved, the current slide number is stored in the source
* A “slideshow open” action can be issued by the user by selecting “File|Open” from the menu. In that case, the application will ask the user to navigate to the stored slideshow by means of a dialog. The system will read the stored slideshow from the file and will navigate to the saved “current slide number”. A slideshow can also be opened by a user by clicking on a slide item which has preconfigured actions attached to it. If one of those attached actions is a “slideshow open” action, the name of the file will have been configured and it will be used to read the slideshow from the file. After reading the slideshow from the file, the system will navigate to the saved “current slide number”
* When the application reads a slideshow from a source (file or other), and an “open slideshow” action is encountered in an action tag, subsequent action tags are ignored, as these additional action tags don’t operate on the same slideshow anymore
* When the application reads a slideshow from a source with name x, and an embedded action in one of the slide items instructs to open the same slideshow with name x, the system will raise an error and not load the slideshow in the embedded action, as this would result in recursive, cyclic “open slideshow ”actions

*Navigational rules*

The user will be able to navigate within the slideshow, browsing through the slides, going to the beginning or the end of the slideshow and navigate directly to a certain slide by providing the slide number in a dialog box. The following rules can be identified (or are assumed if not clearly stated in the u ser-case):

* 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, as the system can deduce the current slide number from the state of the active slideshow
* 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.
* An auxiliary action can only be issued by the system (like a sound) when reading the slideshow from the source. A user cannot issue such action directly (apart from opening a slideshow)

The above rules are summarized in the following CVA table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Type** | **Additional**  **action** | **Activation** |
| go to first, last slide | absolute  navigation |  | * keystroke * mouse-click on slide item * mouse-click in menu |
| go to slide i | absolute  navigation | ask for page through dialog / get page from action in xml | * keystroke * mouse-click on slide item * mouse-click in menu |
| go to next, previous slide | relative navigation |  | * keystroke * mouse-click on slide item * mouse-click in menu |
| open slide | slideshow persistence | ask for source selection by means of dialog | * mouse-click on slide item * mouse-click in menu |
| save slide | slideshow persistence | ask for source selection by means of dialog | * mouse-click in menu |
| auxiliary action | auxiliary action |  | * mouse-click on slide item |

*Table 1: Action rules*

## Assignment 2: Design

This design of the application will be split into the following sections:

* Identification of high-level activities. An activity diagram will be presented that shows the main, high-level activities that are present in the application
* Class diagram of domain model. This class diagram will shed a light on the main entities in the domain model. Not all entities are presented. Only essential entities (abstractions, not implementations in many cases) are depicted to get a good overview of the model
* Class diagram with focus on MVC Design Pattern. Main entities that are involved in the MVC Design Pattern will be included in a separate class diagram
* The concept of action is important in this use-case, and a separate class diagram is shown that depicts the hierarchical structure and other involved entities
* As a good practice, the creation of objects is separated from the usage of the objects. This is reflected in the design, by grouping this facet of the application in a separate sequence of class and interactions diagrams

### High-level activity diagram

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



*Figure x: High-level Activity Diagram*

From the above activity diagram, it can be observed that there are two paths that lead from a user action to either the loading of the slideshow (and the displaying of the “current” slide and its slide items) or the change of a slide (and the displaying of the “current” slide and its slide items):

1. Wait for user input -> load slide show -> draw slide -> draw item (-> draw border)
2. Wait for user input -> draw slide -> draw item (-> draw border)

*In either of those paths, after the initial user input, there is no more user interaction involved. This will be a very important observation that will have its impact in further design decisions.*

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