6 Seaside: Components

These exercises and the ones next week will be using an example of a possible real-world web application. The application is about implementing a ticket box application useable by a theater company having different plays in their program. The application should manage the plays, the shows and the booking of the tickets.

6.1 Introduction

Here we will be starting step by step building up this project. Follow the exercises one by one as they depend on each other.

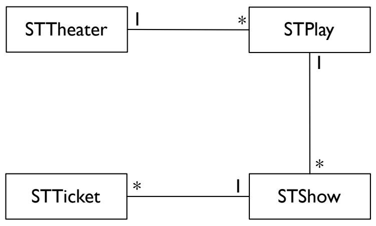


Figure 1: Theater-Model

All the code should be put into the categories *Theater-Model*, *Theater-View* and *Theater-Tests*. The category *Theater-Model* contains a very simple model, as seen in Figure 1, to be used to build up a web-interface around. Feel free to enhance the model when you need to do so, but do run the tests and add new ones to make sure that all the features work as expected after your modifications.

On the class side of `STTheater` you can find a method `#default` returning the domain model to be used for the web application. Usually you do not keep your model just within the image, but use a proper external storage mechanism instead: this can be simply done by dumping out the object graph to the filesystem from time to time or by using a relational- or object-database. However, as possible storage strategies are out of the scope here, we will just keep everything within the image.

Exercise 1 Start out by creating a new task called `STBuyTicketTask` that will model the steps required to buy a ticket. Register it as a new Seaside application as you will need it later on to test your components. Put `self inform: 'Hello World'` into the method `#go` for now. By the end of this section this method should define the flow as seen in Figure 2.

6.2 Choosing a Play

Exercise 2 Create a subclass of `WAComponent` called `STPlayChooser` that will give the user the possibility to choose a theater-play. Add an instance variable `plays` and create accessors to hold a collection of plays that should be displayed

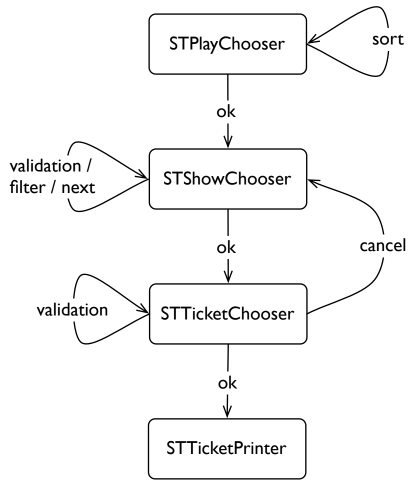
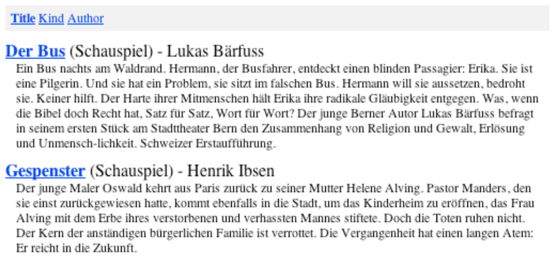


Figure 2: Theater-Flow as defined by `STBuyTicketTask`



Title Kind Author

Der Bus (Schauspiel) - Lukas Bärfuss
Ein Bus nachts am Waldrand. Hermann, der Busfahrer, entdeckt einen blinden Passagier: Erika. Sie ist eine Pilgerin. Und sie hat ein Problem, sie sitzt im falschen Bus. Hermann will sie aussetzen, bedroht sie. Keiner hilft. Der Harte ihrer Mitmenschen hält Erika ihre radikale Gläubigkeit entgegen. Was, wenn die Bibel doch Recht hat, Satz für Satz, Wort für Wort? Der junge Berner Autor Lukas Bärfuss befragt in seinem ersten Stück am Stadttheater Bern den Zusammenhang von Religion und Gewalt, Erlösung und Unmensch-lichkeit. Schweizer Erstaufführung.

Gespenster (Schauspiel) - Henrik Ibsen
Der junge Maler Oswald kehrt aus Paris zurück zu seiner Mutter Helene Alving. Pastor Manders, den sie einst zurückgewiesen hatte, kommt ebenfalls in die Stadt, um das Kinderheim zu eröffnen, das Frau Alving mit dem Erbe ihres verstorbenen und verhassten Mannes stiftete. Doch die Toten ruhen nicht. Der Kern der anständigen bürgerlichen Familie ist verrottet. Die Vergangenheit hat einen langen Atem: Er reicht in die Zukunft.

Figure 3: View of `STPlayChooser`

with this component. Call your newly created component from `STBuyTicketTask`, but don't forget to initialize it with the collection of plays. If you browse to your application, you should get a blank page as you haven't defined any view yet.

Exercise 3 Implement the method `#renderContentOn:`. As a first step, enumerate the plays and display the title of each. If you go back to your web browser and refresh, you should see the titles now. Then display the other information you get from the model. Use your own style sheet or copy the example from Figure 4 to make the output look like Figure 3.

Exercise 4 So far there is no interaction possible with the component. Create an anchor-callback around the title and answer the selected play to the caller. Test your code by extending the task that is calling your component and inform the user about the selected play.

Exercise 5* To set up the list of the plays more convenient, add three links at the top of the page to make it possible to sort the plays according to `#title`, `#kind` or `#author`. To remember the state of the selected sort order you need to

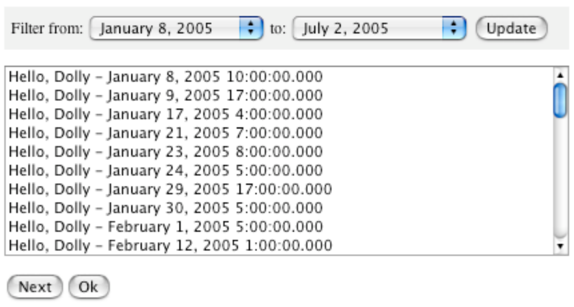
```
.sort {
      background: #eeeeee;
      padding: 5px;
}
.play {
      margin-top: 10px;
}
.play .head {
      font-size: 16pt;
}
.play .body {
      margin-left: 10px;
      width: 490px;
}
```

Figure 4: Stylesheet of `STPlayChooser`

add another instance variable. Make it also possible to sort in reverse order by clicking a second time onto the same link.

6.3 Choosing a Show

Exercise 6 Create another subclass of `WAComponent` called `STShowChooser` that allows the user to choose a show. Add instance variables to hold a collection of shows to choose from and one for the current selection. Create appropriate accessors and call your newly created and properly initialized component from `STBuyTicketTask`.



```
Hello, Dolly - January 8, 2005 10:00:00.000
Hello, Dolly - January 9, 2005 17:00:00.000
Hello, Dolly - January 17, 2005 4:00:00.000
Hello, Dolly - January 21, 2005 7:00:00.000
Hello, Dolly - January 23, 2005 8:00:00.000
Hello, Dolly - January 24, 2005 5:00:00.000
Hello, Dolly - January 29, 2005 17:00:00.000
Hello, Dolly - January 30, 2005 5:00:00.000
Hello, Dolly - February 1, 2005 5:00:00.000
Hello, Dolly - February 12, 2005 1:00:00.000
```

Figure 5: View of `STShowChooser`

Exercise 7 Implement the method `#renderContentOn:` using Figure 5 as a reference; don't worry about the filter yet. Make sure hitting *ok* only answers if the user actually selected a valid show, else show a message that a selection is missing and return to the dialog. Add a button to select the next possible show automatically.

Exercise 8★ Implement a facility to allow filtering for a certain date range. Write a method returning a possible list of dates and add two instance variables to keep the selected date for start and end of the period to be filtered. Render two drop-down boxes and a button to update the filtered list. Use AJAX to update the list of shows without the need to press the update button anymore.

Exercise 9* Experiment with other form controls. How does the interface look like when using option-boxes instead of the list? What do you need to change in the code?

6.4 Buying and Printing Tickets

Exercise 10 Write a component that allows the user to select the number of tickets he wants to buy. Give an error message, if there are not enough places available for the selected show or if the user doesn't enter a valid number. Update the domain model according to the tickets sold and answer a collection of tickets to the task. The view of a minimal implementation can be seen in Figure 6.

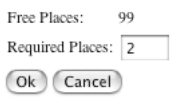
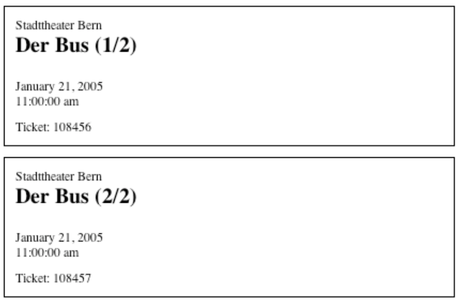

Free Places: 99
Required Places: 2

Figure 6: View of `STTicketChooser`

Exercise 11 Last but not least write yet another component printing out a collection of tickets. This might look like Figure 7. No links or form elements are required in this component. Update your flow accordingly.


Stadttheater Bern
Der Bus (1/2)
January 21, 2005
11:00:00 am
Ticket: 108456

Stadttheater Bern
Der Bus (2/2)
January 21, 2005
11:00:00 am
Ticket: 108457

Figure 7: View of `STTicketPrinter`

Please save the Monticello package `Tutorial` and send it by mail to st-staff@iam.unibe.ch. Attach your written solutions that are not part of the source-code to the mail or hand them in as hardcopy at the beginning of the next exercise session. Your mail and solutions should be clearly marked with names and matrikel numbers of the solution authors.