1

Seaside Tutorial

Main Author(s): lukas renggli

1.1 Getting Started

Follow the instructions given on the slides to install Seaside. Make sure your Seaside server is up and running by accessing the example application at http://localhost:8080/seaside/counter in Squeak or at http://localhost:8008/seaside/go/counter in VisualWorks.

In Squeak load the monticello package tutorial.mcz and in VisualWorks the parcel tutorial.pcl. Both packages contain examples shown during the presentation and some class skeletons that will assist you to do these exercises.

Save your image. From now on work within a copy of this image, so that you can easy go back to a working configuration, in case you severely screw something up.

1.2 Development Tools

Exercise 0 Use your web browser to navigate to the counter example application. Toggle on the halos to see the border of the component this application is built of. Experiment and interact with the application in render- and source-mode.

Exercise 1 Change the behaviour of the increase and decrease buttons: edit the methods #increase and #decrease from within the web browser to increase by 2 and decrese by 3.

Exercise 2 Inspect the living component from within the web browser. There are two instance variables visible, whereas count is representing the state of the component. The other instance variable is defined in a super-class of WACounter and will be discussed later on.

Exercise 3 Change the background color of the web application by using the style editor from within your web browser. Try using something like body { background-color: yellow; }.

Question 4 Why do you think the style editor is used more often in industrial settings than the system browser?

Exercise 5 Introduce an error to the method #increase using your web browser. Play with your application so that the error occurs. Click on the *debug* link which opens a debugger within your image. Fix the bug and proceed the evaluation.

1.3 Control Flow

During the theoretical part an example was shown where the user had to guess a number the computer was thinking of. In this exercise we will have a look at the implementation of two similar games. Some skeletons are provided, so you don't need to implement all by yourself.

1.3.1 User Guesses a Number

Exercise 6 Have a look at the source code of STUserNumberGuesser in the package *Tutorial-Flow* and play the game several times to make sure it works as expected.

Exercise 7 Modify the method #go in STUserNumberGuesser to count the number of guesses. Show the total number of guesses the user required to get the right number in the end of the game.

Question 8 Try using the back button while playing the game. How does the application handle this?

Question 9 What happens if you open multiple windows in the same session and play within the different windows independently?

Question 10★ Is it possible to cheat the counter by using the back button or by opening new windows within the same session? Does this behavior change if you use an instance variable instead of a temporary one for counting?

1.3.2 Computer Guesses a Number

Exercise 11 Write a new web application that allows the computer to guess a number the user is thinking of. In case you run into troubles, you can always have a look at the implementation of STUserNumberGuesser.

1. Create a subclass of WATask called STComputerNumberGuesser.
2. Create an initialization method on the class side of the newly created class, registering the component as a new web application with the path segment cng.
3. Implement the method #go following the rules of the game. Use #inform: to tell the user what he should do and #confirm: to ask the user if the guess of the computer is too big.
4. Play the game several times to make sure it works as expected.

Exercise 12 Implement yet another task asking the user if he wants to guess or not. Depending on the answer either call STUserNumberGuesser or STComputerNumberGuesser. Modify those two classes to answer the numbers of steps required and call them from within your new task. Don't forget to register your new application with a class initialization method.

1.3.3 TicTacToe Game

There are three prepared classes for this game in the package *Tutorial-TicTacToe* following the *MVC-Pattern*:

Model STTicTacToeController is a simple model of a game holding the current board configuration. It includes methods to access and modify its configuration (#boardAt: and #boardAt:put:) and to call an algorithm in order to look for the best possible move of a given player (#find:).

View STTicTacToeView is a simple Seaside view onto the game model. You will learn later on how to create views with Seaside.

Controller STTicTacToeController is a subclass of WATask and this is the place that needs your work now. It already implements a few convenience methods like #newModel, #computerMove and #userMove.

Exercise 13 Register STTicTacToeController as a new web-application, but this time don't use a class initialization method but the configuration interface. Make sure that you have a method #canBeRoot on the class-side so that Seaside recognizes this class as a possible root of a web application. Browse to http://localhost:8080/seaside/config when using Squeak or http://localhost:8008/seaside/go/config when using VisualWorks, enter your password, add a new entry point with the name ttt and select STTicTacToeController as the root component.

Exercise 14 Implement the game in the method #go using the provided convenience methods. You will also need some testing methods of the model to check if the game is finished (#isFinished) and who was the winner (#winner). Don't put all your code into one single method, split it among different ones to ensure readability. Ask the user in the beginning of the game if he prefers to start playing or not.

Exercise 15★ Ensure that the user can't cheat the game by using the back button of the web browser. Don't wrap too much or to few of your code into #isolate: blocks.

1.4 Components

For the rest of this tutorial we will be working on an example of a possible real-world web application: it should be useable by a theater having different plays in its program. The application should manage the plays, the shows and the booking of the tickets.

1.4.1 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. However don't let you hinder from bringing in your own ideas and from implementing some extra features, if you think they could be useful for this project.

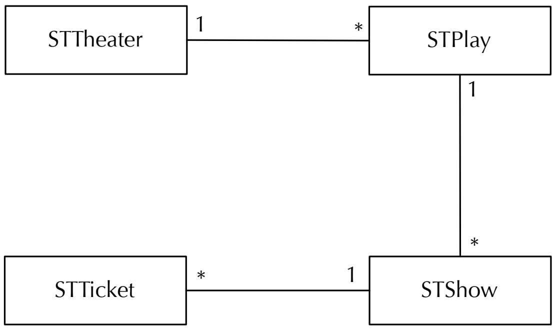


Figure 1.1: Theater-Model

All the code altogether should be put into the bundle *Tutorial-Theater* that contains some packages, namely *Theater-Model*, *Theater-View* and *Theater-Tests*. The package *Theater-Model* contains a very simple model, as seen in Figure **??**, 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 modifictions.

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 16 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. Leave the method #go empty for now. This method should define the flow as seen in Figure **??** by the end of the tutorial.

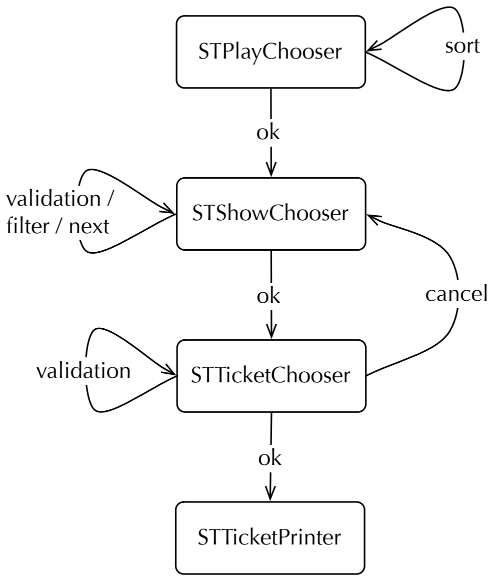
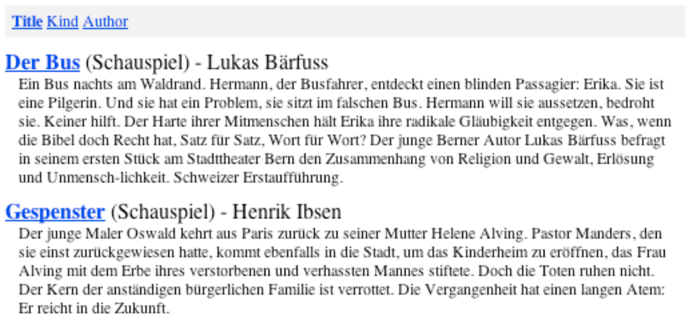


Figure 1.2: Theater-Flow as defined by STBuyTicketTask

1.4.2 Choosing a Play



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 1.3: View of STPlayChooser

Exercise 17 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 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 18 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 **??** to make the output look like Figure **??**.

Exercise 19 So far there is no interaction possible with the component. Create an anchor-callback #anchorWithAction:do: 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 20★ To set up the list of the plays more convenient, add three links at the top of the page to

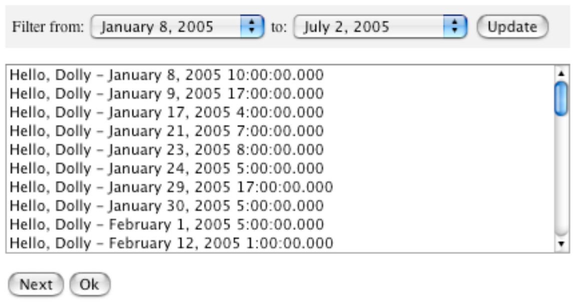
```
.sort {
	background: #eeeeee;
	padding: 5px;
}
.play {
	margin-top: 10px;
}
.play .head {
	font-size: 16pt;
}
.play .body {
	margin-left: 10px;
	width: 490px;
}
```

Figure 1.4: Stylesheet of STPlayChooser

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

1.4.3 Choosing a Show

Exercise 21 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 1.5: View of STShowChooser

Exercise 22 Implement the method #renderContentOn: using Figure **??** 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 23★ 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 live-callbacks to update the list of shows without the need to press the update button anymore.

Exercise 24★ 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?

1.4.4 Buying and Printing Tickets

Exercise 25 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 **??**.

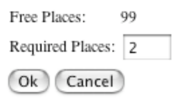
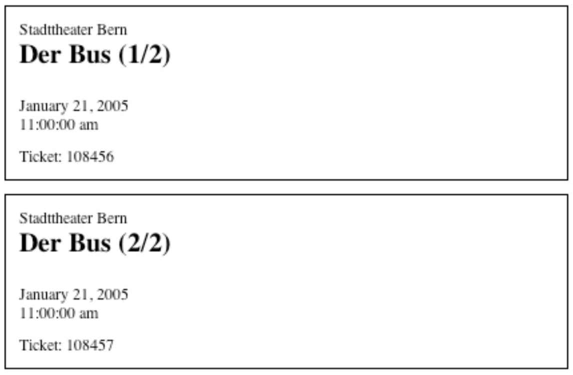


Figure 1.6: View of STTicketChooser

Exercise 26 Last but not least write yet another component printing out a collection of tickets. This might look like Figure **??**. 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 1.7: View of STTicketPrinter

Exercise 27 Make sure that your application implements all the paths that are visible in the state diagram in Figure **??**. Make sure that the user cannot go back after having bought the tickets.

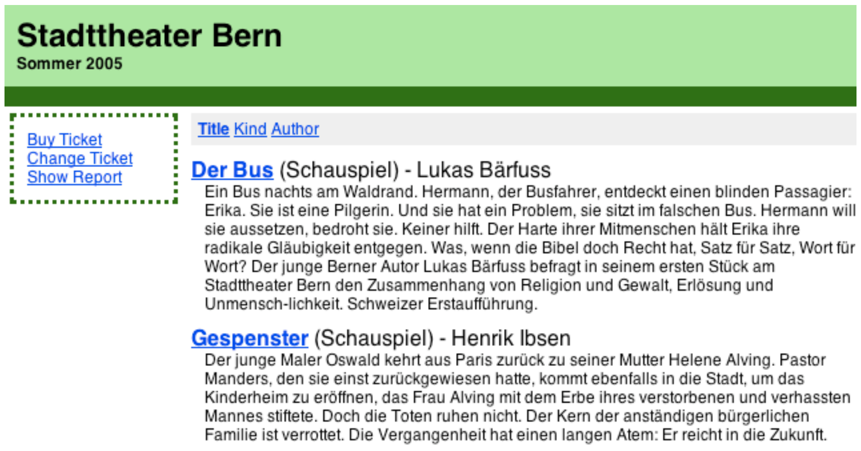
1.5 Composition

In this section we will compose different components we have written before. Create a few more components and plug together an appealing and simple user interface.

1.5.1 Frame, Subcomponent and Backtracking

Exercise 28 Create a new subclass of WAComponent and register it as a new entry point to your application. Render into different div-tags the name of the theater and the current season; you can find this information in the model. Also create a simple menu that is empty for now. Create a style-sheet to make the application look nicer.

Exercise 29 Add an instance variable to your main-frame to hold a child component. Create a method #buyTicket that initializes the variable with a new instance of STBuyTicketTask and send #buyTicket in the initialization method of the component. Place the child beside the menu you have created before. Don't



Stadttheater Bern
Sommer 2005

Buy Ticket
Change Ticket
Show Report

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 1.8: View of STMainFrame

forget to implement the message #children, else you will sooner or later run into troubles. Create a menu item called *Buy Ticket* that sends the message #buyTicket when clicked. Enjoy the application with the halos turned on.

Exercise 30★ Test the new functionality you implemented. Especially try out the behavior of the application when using the back-button. Try clicking on *Buy Ticket*, hit the back-button of your web-browser and then click on any link or control within the child-component. Why do you get an error? Fix the problem and make sure everything works as expected.

1.5.2 Reuse of Components

In this part of the exercises you are basically free about the implementation details of a new requirement of the application: The theater company wants to be able to let the customers return tickets and exchange them with another one from the same play but a different show.

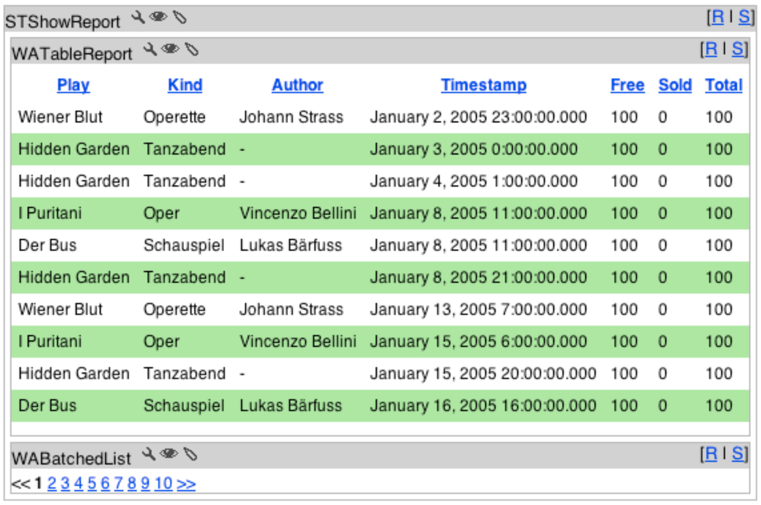
Exercise 31 Use the id of the ticket to identify the one to be replaced. Probably you need to improve the model to make the necessary mutations possible. Also write tests to ensure it works as expected. For the web interface try to write as few lines of code as possible. Reuse the existing components that you have written in the previous steps. You might also want to use components provided by the framework. The example solution requires 7 lines of code, including the validation of the ticket id. Can you do it with less lines of code?

1.5.3 Reporting and Batching

Exercise 32 Create a new component called STShowReport showing a report of all the shows from the model as seen in Figure **??**. Use WABatchedList to enable the batching of the huge list and only display 10 items at once. For the reporting you might want to use WATableReport or write your own component. By default the list should be sorted according to the timestamp. Add the new component to the menu in the main-frame.

1.5.4 Editing a Play

In this part we are going to implement a dialog to edit the attributes of a play. Have a look at Figure **??** to get an idea of the look. Add a link to the title of every play in your report that calls the component you are going to create in the following exercise:


STShowReport [R | S]

WATableReport [R | S]

Play	Kind	Author	Timestamp	Free	Sold	Total
Wiener Blut	Operette	Johann Strass	January 2, 2005 23:00:00.000	100	0	100
Hidden Garden	Tanzabend	-	January 3, 2005 0:00:00.000	100	0	100
Hidden Garden	Tanzabend	-	January 4, 2005 1:00:00.000	100	0	100
I Puritani	Oper	Vincenzo Bellini	January 8, 2005 11:00:00.000	100	0	100
Der Bus	Schauspiel	Lukas Bärfuss	January 8, 2005 11:00:00.000	100	0	100
Hidden Garden	Tanzabend	-	January 8, 2005 21:00:00.000	100	0	100
Wiener Blut	Operette	Johann Strass	January 13, 2005 7:00:00.000	100	0	100
I Puritani	Oper	Vincenzo Bellini	January 15, 2005 6:00:00.000	100	0	100
Hidden Garden	Tanzabend	-	January 15, 2005 20:00:00.000	100	0	100
Der Bus	Schauspiel	Lukas Bärfuss	January 16, 2005 16:00:00.000	100	0	100

WABatchedList [R | S]

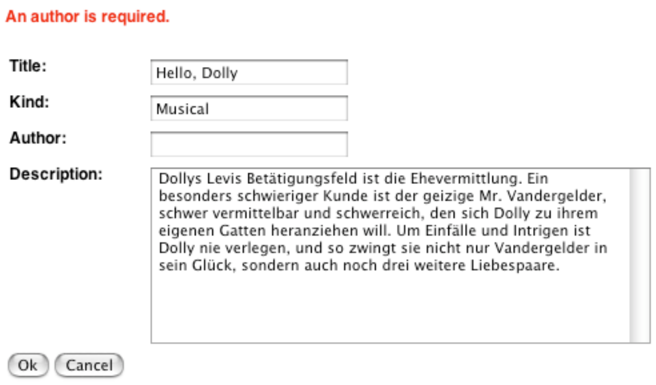
<< 1 2 3 4 5 6 7 8 9 10 >>

Figure 1.9: View of STShowReport with halos toggled on

Exercise 33 Create a new subclass of WAComponent and add an instance variable to hold the play. In the method #initialize wrap the component with two decorations:

1. WAFromDecoration to render a form around the component and display *ok* and *cancel* buttons.
2. WAValidationDecoration to validate the input fields and display an error message if necessary.

Ensure that the validation errors are properly displayed and that the model isn't touched when hitting cancel.


An author is required.

Title:	Hello, Dolly
Kind:	Musical
Author:	
Description:	Dollys Levis Betätigungsfeld ist die Ehevermittlung. Ein besonders schwieriger Kunde ist der geizige Mr. Vandergelder, schwer vermittelbar und schwerreich, den sich Dolly zu ihrem eigenen Gatten heranziehen will. Um Einfälle und Intrigen ist Dolly nie verlegen, und so zwingt sie nicht nur Vandergelder in sein Glück, sondern auch noch drei weitere Liebespaare.

Figure 1.10: View of STEditPlay

Exercise 34★ Load Mewa and try to write the same dialog using a descriptive meta model.

1.6 Advanced

1.6.1 Continuations

To answer the following question it might be useful to have a look at the class Continuation. You might also want to run the different tests of ContinuationTest and type and evaluate a few expressions in the workspace.

Question 35 When should one *not* use a continuation based web framework?

Question 36 How are continuations implemented in Smalltalk? Why are there no primitives required?

Question 37 What about the time- and space-performance of continuations?

Question 38★ Why is the implementation of the class Continuation polymorphic to BlockClosure? What are the differences?

Question 39★ When are ensure-blocks evaluated, if you create a continuation within a protected context?

1.6.2 Bookmark-able URLs

Exercise 40 Implement #updateUrl: in the three top-level sub-components of your web application and add an appropriate path-element to the URL. Depending on the context of your application, the URL should now look like: ../theater/buy, ../theater/change and ../theater/report.

Exercise 41 So far it isn't possible to navigate to these sub-components directly using an URL. To get the desired result, create a subclass of WARenderLoopMain called STRenderLoopMain and override the message #start: to parse the URL and to setup the root component as requested.