Magritte Tutorial

Lukas Renggli
renggli@iam.unibe.ch
www.lukas-renggli.ch

Software Composition Group
Institut f¨ur Informatik und angewandte Mathematik
University of Bern, Switzerland

December 1, 2006

1 Getting Started

Fire up the prepared image and get yourself ready for the exercises. Make sure your Seaside server is running within the image by browsing the counter application at http://localhost:8080/seaside/counter.

The image you are using has been created from a fresh Squeak 3.9 image, by loading the following packages from SqueakMap: Smalltalk, eCompletion and Magritte. Also a package called Magritte-Tutorial has been added, where you can put your code.

You can easily create your own development image by taking a fresh 3.7, 3.8 or 3.9 image and loading Magritte from SqueakMap yourself, note that your computer will be busy for a while because this will pull in a few other prerequisites (such as a Web server and Seaside).

The exercises start simple and with detailed instructions on how to perform the given tasks. Exercises marked with a star are a bit trickier, you might want to solve them later on. Sometimes you will probably not exactly know what class to use, what method to call or what parameters to pass, use the power of Smalltalk (senders, implementers, references, ...) to browse the source-code of Magritte, you might even discover some other features that were not presented during the lecture.

2 Juggle with Descriptions

Here a look at the source-code of the classes MAPersonModel, MAAddressModel, and MAModel, that have been presented during the lecture and that are pre-installed in your image. Browse to http://localhost:8080/seaside/person-editor and check if you can see all the features presented during the lecture. All the following exercises will be built upon this simple model.

Exercise 1 Add a new field that holds a Comment about the person, display it as a text-area field as the last element. Test it in the Web browser by starting a new session on the same application. If you are required to add more than one new method by hand you did something wrong.

Solution 1 In the MADescription hierarchy there is a class MAStringDescription which is used to describe a string. You can try to use it like this:

```
MAUserModel subclass: descriptionComment

MAStringDescription new: 'comment' label: 'Comment' priority: 100
```

The form input rendered is a text-input wherein the subject asks you for a text-area (a text-input on more than one line). Have a look at MAStringDescription subclasses. You will see a MAEmailDescription class with a #isValid method. Use this class:

```
MAUserModel subclass: descriptionEmail

MAStringDescription new: 'email' label: 'Email' priority: 100
```

A text-area is then displayed.

Exercise 2 Add a new field that holds the Nationality of the person, display it as a sorted drop-down box with a few selectable countries. Put it right after the Address field. The default choice for new objects should be Switzerland.

Solution 2 When you want to ask the user to choose one element in a list of multiple, you should use the MAStringOptionDescription class. This class is a subclass of MAOptionDescription which accepts an #options: method to specify the different choices. The default choice [displayed by magritte if no other choices have been selected] is chosen with the #default: method:

```
MAUserModel subclass: descriptionNationality

MAStringOptionDescription new: 'nationality' label: 'Nationality' priority: 50

options @ ['Switzerland', 'France', 'Germany']

default: 'Switzerland'

isRequired: true

```

You can use the #sortBy method to sort values in the list.

Exercise 3 Add a new field that holds the E-Mail of the person, display it as a required text-field. Add custom conditions to force the user to give a valid e-mail address. Don’t allow addresses from the providers hotmail.com and gmx.com, gmx.de, gmx.it, etc. Test your code in the Web browser.

Solution 3 An E-Mail is basically a string. So, to start, you can just ask for a string:

```
MAUserModel subclass: descriptionEmail

MAStringDescription new: 'email' label: 'Email' priority: 100

isRequired: true

```

If you open a browser on MAStringDescription class (protocol validation), you will see a #isValidCondition:labelled: method.
Exercise 4* Reuse the description of the `EmailAddress` from `MAddressModel` in the model of the address `MAddressModel` by calling the appropriate description of the person and changing the label to `Email`. Make sure not to modify the original description by creating a copy. Test it in the Web browser.

Solution 4 Descriptions are created in methods. To reuse a description, you can just send the method:

```javascript
MAddressModel class>>description

^ aMzStringDescription autos 'email' labels 'Email'
```

Question 5 Make a rough guess on how much code was saved by using Magritte compared to a manual approach in Seaside. From when on does it make sense to use a meta-model? What are the advantages? What are the disadvantages?

3 Integration into Seaside

Up to now we have been working with descriptions only, we didn’t write any line of Seaside code. Moreover the model was not remembered somewhere and therefore was lost between different sessions. In this section we will concentrate on Seaside and build a simple user-interface that allows us to manage multiple persons. To get an idea of how the result could look like, see Figure 1.

**Person Manager**

1. Lukas Renggli view
2. Stephanie Dogass view

**Figure 1: Person Manager List**

Exercise 6 Start by creating a new sub-class of `MComponent` called `MPersonManager`. Add a class-instance-variable called `persons` that will serve as a simple place to keep the model objects. Initialize it with an empty `OrderedCollection`. Register the newly created class as a new Seaside entry point.

Create a method `renderContentOn:` that displays the heading `Person Manager`. Test the setup of your new application in the Web browser.

Solution 6 Create this methods:

```javascript
MPersonManager class>>person

Persons

MPersonManager class>>addCollection

Persons = addCollection

MPersonManager class>>initialize

super initialize

self register:

self register:

application 'personManager'

MPersonManager>>renderContentOn:

html html header 'Person Manager'

...
Exercise 9  Render the list entries as links, so that existing persons can be edited.

Solution 9

```html
MzPersonManager#viewPerson: aPerson
self call: (MzPerson multiComponent update: #cancel)
readly: #view
renderLineOfPerson: aPerson on: html
total anchor
  onclick: [ self editPerson: aPerson ];
  with: [ html text: aPerson firstName, space, text: aPerson lastName ]
```

Exercise 10  Render an additional link called `view`, as seen in Figure 1, to display the entry in a read-only view. You can turn a component into read-only mode by sending the message `#readonly:`.

Solution 10

```html
MzPersonManager#viewPerson: aPerson
self call: (MzPerson multiComponent update: #cancel)
readly: #view
renderLineOfPerson: aPerson on: html
total anchor
  onclick: [ self editPerson: aPerson ];
  with: [ html text: aPerson firstName, space, text: aPerson lastName ]
```

4 Using Magritte Reports

Rendering a list like this is nice, but it doesn’t scale well if you have many entries. As well it doesn’t take advantage of the nice descriptive model we have. Luckily Magritte includes built in reporting facilities that we will be extending our application with. In this section we make our little application look like Figure 2.

Exercise 11  Add a new child component to MzPersonManager and initialize it, lastly or within the method `initialize`, with `MzReport` rows: self persons description: MzPersonModel description. Render the report instead of the list.

Solution 11

```html
MzPersonManager#report
  report
```

Person Manager

Add Filter Clear
First Name Last Name E-Mail
Nationality

<table>
<thead>
<tr>
<th>Lukas</th>
<th>Rennholz</th>
<th><a href="mailto:rennholz@iwm.unibe.ch">rennholz@iwm.unibe.ch</a></th>
<th>Switzerland</th>
<th>view edit remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sophie</td>
<td>Ducasse</td>
<td><a href="mailto:ducasse@iwm.unibe.ch">ducasse@iwm.unibe.ch</a></td>
<td>France</td>
<td>view edit remove</td>
</tr>
</tbody>
</table>

Figure 2: Person Manager Report

```html
MzPersonManager#report: aReport
  report := aReport
MzPersonManager#edit:
  (MzReportCollection with: self report)
MzPersonManager#initialize
  super initialize
MzPersonManager#renderContentOn: html
  html heading: 'Person Manager'
  html anchor css: #add of: self
  html render: self report
```

Exercise 12  Play with the report in your browser: sort the rows, add new persons... You will probably notice that the report doesn’t update itself when adding new persons. It’s because the report creates a copy. Call `#refresh` to do so.

Solution 12

```html
MzPersonManager#add
  | person |
  | person := self call: (MzPerson multiComponent update: #refresh)
  | render:
  |  addPerson
  |PersonsSelection
  |#add
  |self class persons add: person.
  |self report refresh
```

Exercise 13  Right now almost all the information about a person is displayed within the report, if you enter much data this gives a very wide Web page. Filter out all the descriptions except the ones for `#firstName`, `#lastName`, `#email`, and `#nationality`.

Solution 13

```html
MzPersonManager#initialize
  super initialize
  self report := MzReport
    rows: self persons description select: [ #firstName lastName email nationality]
    includes: #add accessor selector]]
```

Exercise 14  Add a new column to the report to hold commands to view, `edit` and `remove` entries. To get a hint on how to archive the desired behaviour, browse the references of `MCommandColumn`.

```html
```
Solution 14

```ruby
MzPersonManager\>initialize
super initialize
self reports aMzReport
rows (\#description: \#result: \#matches: \#width: and \#actions: \#renderer: \#width: \#classifier: \#report: \#selector: \#all:)
```

Exercise 15* Implement a search functionality within your report. The user should be asked for a filter string that will be compared to all the string-fields within the persons. To get some pointers there have a look at the implementers and senders of \#renderer: \#classifier: \#description: \#result: \#matches: \#actions: \#renderer: \#width: \#classifier: \#report: \#selector: \#all: Add a clear link as well, to remove any filter.

Solution 15

```ruby
MzPersonManager\>filter
\{ filter \}
```

Exercise 16* Point your browser to http://localhost:8080/seaside/example-browser and select MADescriptionEditor in the first drop-down box. What is this? What could it be useful for?

Exercise 17* Browse the source code of MADescriptionEditor. As you can see for the preview and instance of MAScaffolder is created. This is a class that has only two instance-variables: one references a description-container, the other a dictionary mapping description-elements to actual values. Have a look at the methods \#Renderer: \#classifier: \#description: \#result: \#matches: \#actions: \#renderer: \#width: \#classifier: \#report: \#selector: \#all: Why are these methods overridden in MAScaffolder?

5 Described Descriptions*

As experience shows, customers are often not completely satisfied with the features the application developers are giving to them. They want more and they want to be able to customize everything by themselves. Thanks to the reflective nature of Magritte – all the descriptions within Magritte are described with Magritte descriptions itself – we can easily provide the necessary functionality.

Exercise 18* Add a class-instance-variable to MAPersonModel called CustomDescription. Initialize it with an empty instance of MContainer. Create an accessor method. Override \#description on the instance-side, call super and compose it with the CustomDescription. So far, you should see no difference in the behaviour of the application when testing it.

Solution 18

```ruby
MzPersonModel\>class\>description
\{ CustomDescription \}
```

Exercise 19* Add another link above your report to let the user edit the custom description. Unfortunately the default implementation of MADescriptionEditor has no way to go back to the caller. The simplest solution is to subclass MADescriptionEditor and render an additional button by overriding \#renderer: \#all: \#renderer: \#width: \#classifier: \#report: \#selector: \#all: that answers the modified description.

Solution 19

```ruby
MzPersonModel\>renderer\>description
\{ MADescriptionEditor\>description \}
```

Exercise 20* Now you are able to add additional fields to your MPersonModel class, unfortunately the editor doesn’t work anymore. If you try to add or edit a person you get the error-message: This message is not appropriate for this
object. Obviously the custom-descriptions that have been added through the 
web do not have an accessor and you probably don’t want to let your customers 
care about those low-level things anyway. To solve the problem you need to 
override the methods #reading: and #write: using; similar to the way it is 
done in MAScaffold.

Solution 20 Add an instance variable values into MAPersonModel.

MAPersonModel>>values
  ^ value (self) { values = Dictionary new }

MAPersonModel#reading: aDescription
  " [MAPersonModel customDescription includes: aDescription]
  self: (self values at: aDescription description: nil)
  self: [super reading: aDescription]

MAPersonModel#writing: self: aDescription
  " [super write: self: using: aDescription]

Question 21 Test your application toughly. How does it behave if you add, 
remove or change fields within living instances of MAPersonModel?

Edit Person

<table>
<thead>
<tr>
<th>Title:</th>
<th>Mr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name:</td>
<td>Lukas</td>
</tr>
<tr>
<td>Last Name:</td>
<td>Renggli</td>
</tr>
<tr>
<td>E-Mail:</td>
<td><a href="mailto:renggli@tum.unibe.ch">renggli@tum.unibe.ch</a></td>
</tr>
<tr>
<td>Home Address:</td>
<td>Create</td>
</tr>
<tr>
<td>Office Address:</td>
<td>Create</td>
</tr>
<tr>
<td>Nationality:</td>
<td>Switzerland</td>
</tr>
<tr>
<td>Picture:</td>
<td>Choose file</td>
</tr>
<tr>
<td>Birthday:</td>
<td>6 June 1980</td>
</tr>
<tr>
<td>Age:</td>
<td>26</td>
</tr>
<tr>
<td>Kind:</td>
<td>Number</td>
</tr>
<tr>
<td>Phone Numbers:</td>
<td>The report is empty.</td>
</tr>
<tr>
<td>Add</td>
<td></td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
</tr>
<tr>
<td>Custom Field 1:</td>
<td>Too hot</td>
</tr>
<tr>
<td>Custom Field 2:</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Person Editor Customized