User's Handbook

map.apps with MD2

Project Seminar

Model-driven Mobile Development

(University of Münster)

23rd February 2015

Contents

1 Introduction

2 Modeler's Handbook

2.1 Installation

The following software is required prior to the next steps:

- map.apps 3.1.0
- Eclipse IDE for Java and DSL Developers (e.g. Version Kepler)
- NetBeans EE (e.g. Version 8.0)
- Apache Tomcat 7.0 with running map.apps runtime
- GlassFish 4.+

2.2 Getting Started

2.2.1 Developing a Single App

In this section it is described how an application can be developed based on the current (March 2015) state of the MD2 DSL. To this end, this section is structured according to the different layers which constitute MD^2 model and are represented in Figure 2.1. Aside from the conventional MVC layers – model, view and controller – this includes an additional layer, which enables and models the generation of workflows within and across apps and was established during the project seminar.

All components in MD^2 are organized in a package structure that represents the aforementioned structure. All documents have to be placed in corresponding packages (views, models, controllers or workflow). For example, all view files are expected to be in the package any.project.package.views. The package has to be defined in each MD^2 file as follows:

```
package PACKAGE_NAME
```

The package name has to be a fully qualified name that reflects the actual folder structure.

Workflow

The workflow layer is an additional abstraction on top of the controller layer. It thereby allows to specify the general course of action of one or more apps with few simple and well understandable model

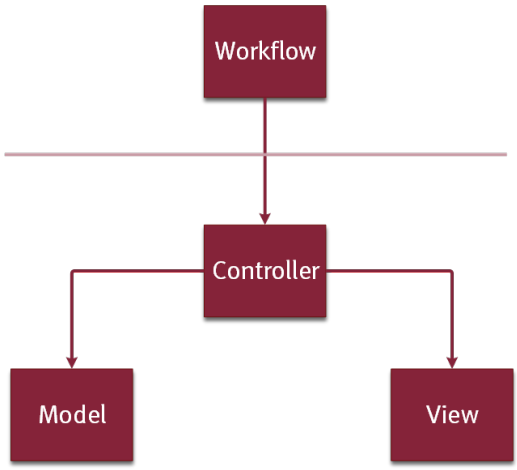

Figure 2.1: Architecture of MD^2 Models

language constructs. Furthermore, this abstract workflow representation is intended to serve as a basis for communication with customers, e.g. for requirements engineering and collaborative app development.

In the workflow file of an MD2 model a workflow can be specified as a (possibly cyclic) directed graph of workflow elements. Workflow elements represent encapsulated functionality which is specified in detail in the controller layer. The workflow layer references the workflow elements from the controller layer to define their interaction.

For this purpose, Workflow elements are linked via events. For each workflow element one or more events can be specified that can be fired. However, at runtime a workflow element can fire only one of these events, i.e. a parallel processing of the workflow is not intended. In addition to the events that can be fired, the workflow element also specifies which workflow element is to be started in response to a fired event using the keyword `start`. A workflow element in the workflow layer typically looks as shown in Listing 2.1.

Listing 2.1: Workflow Elements in the Workflow Layer

```
WorkflowElement NameOfWorkflowElement
      fires NameOfEventOne {
            start NameOfSubsequentWfeOne
      }
      fires NameOfEventTwo {
            start NameOfSubsequentWfe
      }
```

After defining the sequence of workflow elements, the workflow also requires the specification of an app. As shown in Listing 2.2, an app consists of its ID, a list of workflow elements that are used in the app and a name that is to be used as app titel. For the scenario where only a single app is modeled, all workflow elements can be listed in the app. However, it is also possible to have unused workflow elements.

Listing 2.2: App Definition in MD2

```
App AppID {
      WorkflowElements {
            WorkflowElementOne,
            WorkflowElementTwo (startable: "Start Workflow Element Two"),
            WorkflowElementThree
      }
      appName "App Title"
}
```

A workflow has one or more entry points, i.e. startable workflow elements. These are marked as `startable` in the app specification. During code generation this will result in a button within the app that starts the corresponding workflow element. In addition, a string needs to be inserted which is used as label or description for the button.

Finally, the complete workflow specification for one app will be structured as shown in Listing 2.3. Note that MD2 does not differentiate between different workflows. However, it is possible to implicitly create multiple workflows by using two or more startable workflow elements that start independent, disjunct sequences of workflow elements.

Listing 2.3: Workflow Definition in MD2

```
package ProjectName.workflows

WorkflowElement WorkflowElementOne
[...]
WorkflowElement WorkflowElementTwo
[...]
WorkflowElement WorkflowElementThree
[...]

App AppID {
      [...]
}
```

Model

In the model layer the structure of data objects is being described. As model elements Entities and Enums are supported.

Entity An entity is indicated by the keyword `entity` followed by an arbitrary name that identifies it.

```
entity NAME {
```

```
<attribute1 ... attribute n>
}
```

Each entity may contain an arbitrary number of attributes of the form

```
ATTRIBUTE_NAME: <datatype>[] (<parameters>) {
name STRING
description STRING
}
```

The optional square brackets [] indicate a one-to-many relationship. That means that the corresponding object may hold an arbitrary number of values of the given datatype. Supported complex data types are:

- Entity
- Enum

Supported simple data types are:

- `integer` – integer
- `float` – float of the form #.#
- `boolean` – boolean
- `string` – a string that is embraced by single quotes (') or double quotes (")
- `date` – a date is a string that conforms the following format: `YYYY-MM-DD`
- `time` – a time is a string that conforms the following format: `hh:mm:ss[(+|-)hh[:mm]]`
- `datetime` – a date time is a string that conforms the following format: `YYYY-MM-DDThh:mm:ss [(+|-)hh[:mm]]`

Parameters are optional and will be transformed into implicit validators during the generation process. They have to be specified as a comma-separated list. On default each specified attribute is mandatory. To allow null values the parameter optional can be set. Further supported parameters depend on the used data type and are explained as follows:

- `integer` supports
  - `max INTEGER` – maximum allowed value of the attribute
  - `min INTEGER` – minimum allowed value of the attribute
- `float` supports
  - `max FLOAT` – maximum allowed value of the attribute
  - `min FLOAT` – minimum allowed value of the attribute
- `string` supports
  - `maxLength INTEGER` – maximal length of the string value
  - `minLingth INTEGER` – minimal length of the string value

Optionally, attributes can be annotated with a name and a description which are used for the labels and the tooltips in the auto-generation of views. If a tooltip is annotated a question mark will be shown next

to the generated input field. If no name is annotated, a standard text for the label will be derived from the attribute's name by transforming the camel case name to natural language. E.g. the implicit label text of the attribute firstName is "First name".

Exemplary entity that represents a person:

```
entity Person {
	name: string
	birthdate: date {
		name: "Date of Birth"
		description: "The exact day of birth of this person."
	}
	salary: float (optional, min 8.50, max 1000)
	addresses: Address[]
}
```

Enum An enumeration is indicated by the keyword `enum` followed by an arbitrary name that identifies it. Each enum may contain an arbitrary number of comma-separated strings. Other data types are not supported. Exemplary enum element to specify weekdays:

```
enum Weekday {
"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
}
```

View

Controller

2.2.2 Deploying a Single App

Backend

map.apps

2.3 Development and Deployment of Multiple Apps

2.3.1 Workflow across multiple Apps

2.3.2 Deployment on map.apps

2.4 Additional Features

2.4.1 Uploading, Saving and Displaying RESTful Web Services

2.4.2 Calling RESTful Web Service from the App

2.4.3 Control of Workflow by calling a RESTful Web Service

3 Developer’s Handbook

A Sample Workflow