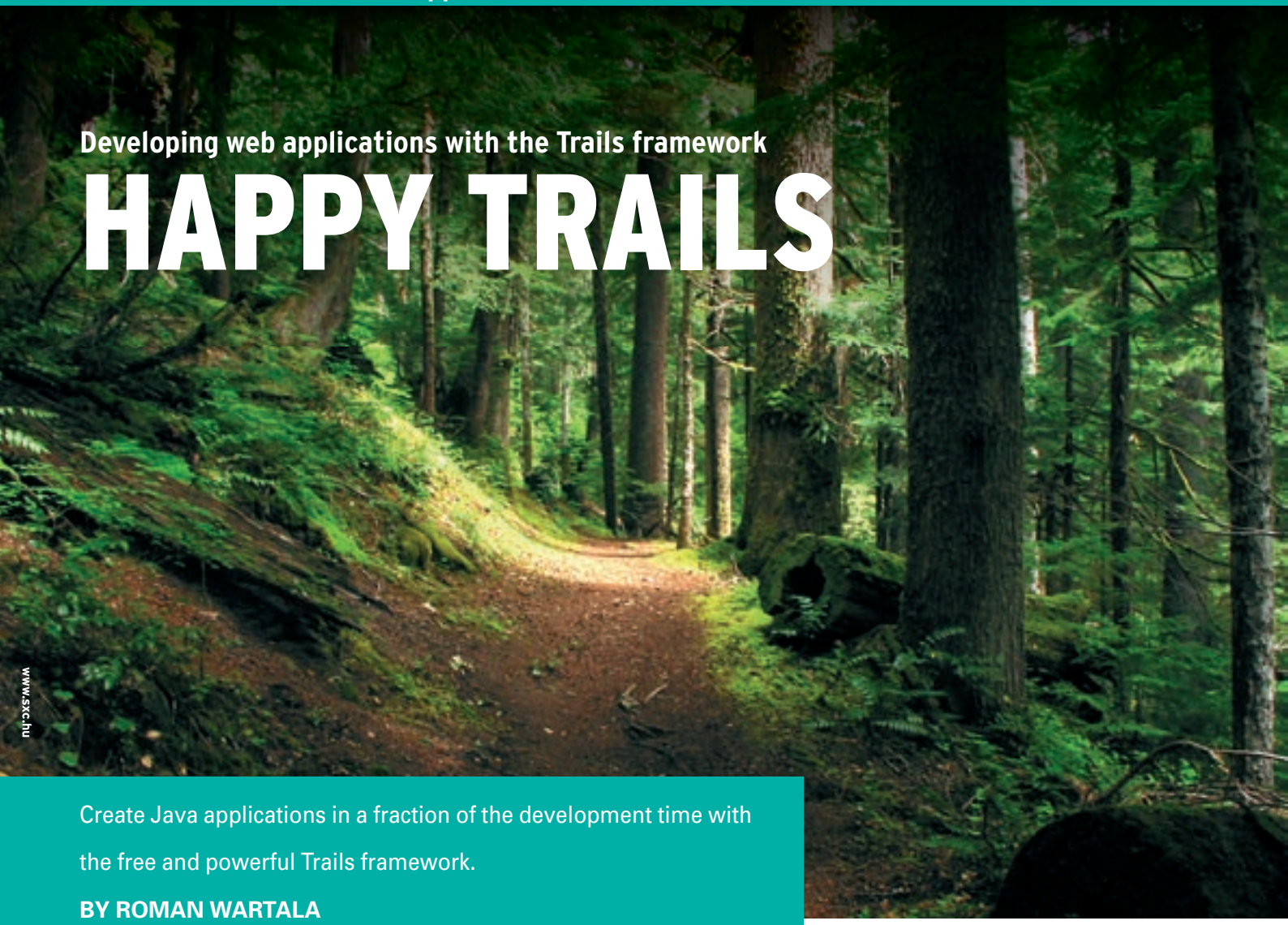

Developing web applications with the Trails framework

HAPPY TRAILS

www.sxc.hu

Create Java applications in a fraction of the development time with the free and powerful Trails framework.

BY ROMAN WARTALA

It must have been sometime early in the summer of 2005 when Jim Weirich showed his friend Chris Nelson the Ruby on Rails [1] video at a meeting of the Cincinnati Java User Group. Chris was impressed by the simple and quick approach to developing web applications and decided to introduce the same approach to his favorite programing language, Java. A few months later, Chris came up with the goods, introducing the first version of Trails [2], an elegant web framework for Java.

The purpose of the Trails project is to "…make Java enterprise development radically simpler by allowing developers to focus on the domain model" while other portions of the code are dynamically generated [1]. Trails minimizes the quantity of original coding and automatically generates as much of the code as possible. The name *Trails* is derived from *Rails* and adds a *T* for *Tapestry* [3]. Tapestry is an open source framework for building web applications in Java. Trails also relies on concepts and components from other trusted frameworks, such as Apache Ant, AspectJ [4], Spring [5], and the object relational mapper Hibernate.

In this article, I'll put Trails to work on a small example application: a tool that manages information on videos.

Quickstart

To work with Trails, you need to install Ant and the 20Mbyte Trails archive from [2]; the archive includes all the other frameworks you need to run Trails. You should also have Tomcat 5.5 running as your application server to avoid delays. After setting *ANT_HOME* and unpacking Trails in an appropriate directory, you can give the *ant install-apt* command to install the required Ant library for Trails in the *Ant-Lib* directory.

Trails automatically generates the skeleton application (Rails refers to this as the scaffolding) and populates it with the required libraries and configuration files. To do this, all you need to do is run *ant create-project* against the Trails directory .

After entering the root directory and a name for the new application (*myvideos*, in this example), Trails creates a new directory along with the required subdirectories and libraries, including a *build.xml* file for creating and deploying the new web application. The directory structure for this new directory is as follows:

- *Root directory/myvideos/*: Main directory for the web application. This directory can be used as the starting point for importing a project into an IDE such as Eclipse.
- *Root directory/myvideos/src*: Source code directory for the application. This is where you implement your own classes.
- *Root directory/myvideos/context*: Directory for the web application.
- *Root directory/myvideos/context/WEB-INF*: This directory houses the required configuration files (web.xml, hibernate.properties, and so on), along with the Tapestry pages and HTML fragments.

- *Root directory/myvideos/lib*: A directory for all archives with various frameworks (Hibernate, Tapestry, Apache Commons, and so on).

You also need to specify the path to your Tomcat installation in *build.properties*.

Domain-based Development

The starting point for any Trails web application is one or more POJOs (Plain Old Java Objects), which map to the domain objects in the application. This is the first major and critical departure for those familiar with Ruby on Rails: Trails does not start with a database table, but expects a simple Java class as its starting point.

The class in Listing 1, which manages the videos in our sample application, is easy to create using a modern IDE such as Eclipse. The programmer simply needs to declare the class attributes *id*, *title*, and *year*, and then use the getters and setters generator (Figure 1) to create the appropriate access methods.

To tell Hibernate which class to persist, developers need to use Java 5 annotations that comply with the new EJB-3.0/JSR 220 standards [6][7]. The *@Entity* annotation (Listing 1) specifies that Hibernate should store the *Movie* class in the database. A unique ID number is still missing: the annotation *@Id(generat e=GeneratorType.AUTO)* creates the ID using the *id* attribute to do so.

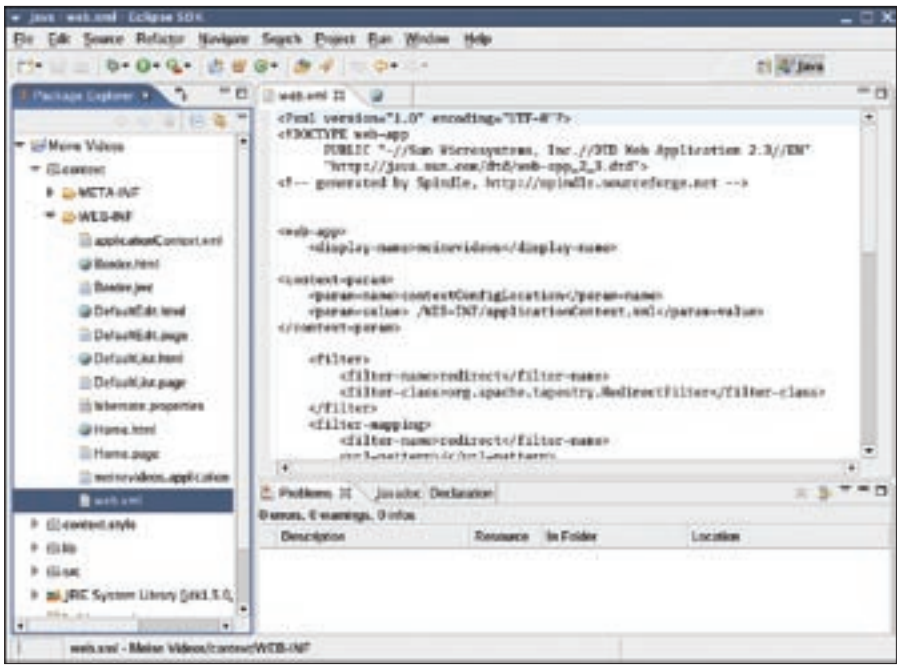

Figure 1: You can use the getters and setters generator to create the appropriate access methods.

To be able to distinguish between the movies in the database, we also need an *equals*, which is trivially implemented using an *EqualsBuilder* from the *apache.commons.lang.builder* framework. All we need to do now is call the Ant target, *ant deploy*; this sends the web application to the address *http://localhost:8080/myvideos/*.

CRUD

The welcome page of the video application wishes users a friendly "Welcome." Clicking on *List Movies* takes you to the first surprise, although you may be expecting this if you have worked with

Listing 1: Movie (1)

```
01 package de.wartala.myvideos;
02
03 import java.util.HashSet;
04 import java.util.Set;
05
06 import javax.persistence.
   Entity;
07 import javax.persistence.
   GeneratorType;
08 import javax.persistence.Id;
09
10 import org.apache.commons.
   lang.builder.EqualsBuilder;
11
12 @Entity
13 public class Movie {
14     private Integer id;
15     private String title;
16     private Integer year;
17
18     @Id(generate=GeneratorType
   .AUTO)
19     public Integer getId() {
20         return this.id;
21     }
22
23     public void setId(Integer
   id) {
24         this.id = id;
25     }
26
27     public String getTitle() {
28         return title;
29     }
30
31     public void setTitle(String
   title) {
32         this.title = title;
33     }
34
35     public Integer getYear() {
36         return year;
37     }
38
39     public void setYear(Integer
   year) {
40         this.year = year;
41     }
42
43     public boolean
   equals(Object obj) {
44         return EqualsBuilder.
   reflectionEquals(this, obj);
45     }
46
47     public String toString() {
48         return this.getTitle();
49     }
50 }
```

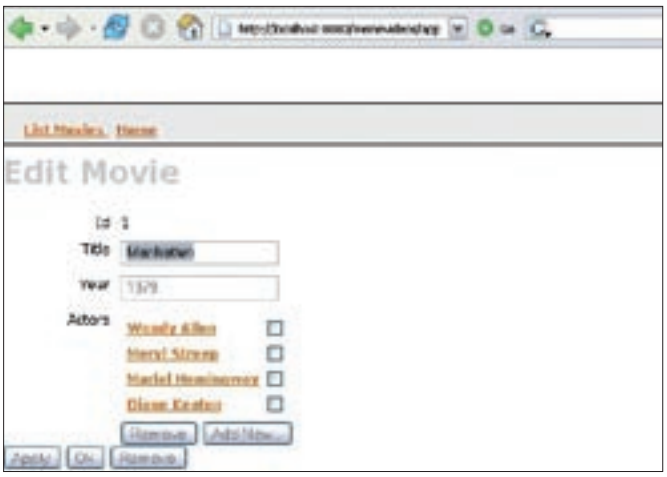

Figure 2: Trails lets you develop Java web applications like this movie database minimal manual coding.

Rails: even at this early stage of development, users can easily create, search, modify, or delete movie records. This design is known as CRUD (Create, Retrieve, Update, Delete), and is one of the highlights of any Rails demonstration. And, hey presto! Clicking on *New Movie* pops up an input form for the first movie record. A simple Trails function, the so-called Pluralizer, automatically generates the plural *Movies* from the *Movie* POJO; simply pass a class name in the singular to the pluralizer to generate the plural.

We want our sample application to manage actors as well as movies. To do this, we need to implement a matching domain object. The *Actor* class will contain the attributes Name and Birthday (Listing 2). Hibernate wouldn't be an object relational wrapper if it was only capable of mapping objects. Of course it can express relationships between individual entities. In the movie management application, we need the ability to assign multiple actors to a single movie, a classical 1 to n relationship. To express this within the *Movie* class, you can use Java 5's Generics.

A hashset of *Actor* implements the 1 to n relationship for the actors in the *Movie* class (Listing 3, Line 25). Getters and setters for this class attribute use JSR 220 annotations to express the required relation (Listing 3, Lines 57 through 68). *ant redeploy* applies the changes. Now, a user who enters a movie can click *Add New* to add actors.

Your Own Website

The visible websites thus far have been generated by Trails and are fairly plain. Trail annotations give developers the ability to make simple changes, such as modifying the input box order or the attribute titles. For example, the following annotation sets the position of the actor birthday field in the form (Position 2), the output format for the date, and the label.

```
@PropertyDescriptor(index=2,➥
format="dd.MM.yyyy",➥
displayName="Birthday")
public Date getBirthday() {
        return birthday;
}
```

Of course, modifications of this type should not reside within the class but outside the source code. To do this, the pages themselves have to be parametrized.

To support structural changes to an underlying page, Trails uses the Tapestry web framework for model view controllers. Tapestry supports simple implementation of website validation and internationalization. Just like Java Server Faces (JSF), Tapestry distinguishes between the content components on a page and their representation. All the input elements shown in in Trails are Tapestry or Trails components. If you want to change the generated pages, you need to concern yourself with the basic distribution of the individual elements in Tapestry. Trails generates default pages for CRUD cases (*.html* extension) and page models to match (*.page* extension). Both

Listing 2: Actor

```
01 package de.wartala.myvideos;
02
03 import java.util.Date;
04
05 import javax.persistence.
   Entity;
06 import javax.persistence.
   GeneratorType;
07 import javax.persistence.Id;
08
09 @Entity
10 public class Actor {
11     private Integer id;
12     private String name;
13     private Date birthday;
14
15     @Id(generate=GeneratorType
   .AUTO)
16     public Integer getId() {
17         return id;
18     }
19
20     public void setId(Integer
   id) {
21         this.id = id;
22     }
23
24     public String getName() {
25         return name;
26     }
27
28     public void setName(String
   name) {
29         this.name = name;
30     }
31
32     public Date getBirthday()
   {
33         return birthday;
34     }
35
36     public void
   setBirthday(Date birthday) {
37         this.birthday =
   birthday;
38     }
39
40     public String toString() {
41         return this.getName();
42     }
43 }
```

parts are controlled by the Tapestry application servlet, which is integrated via the *web.xml* file.

Localization

If you want to localize static page content, all you need to do is to change the corresponding HTML pages. This is where Java-style message bundles are used. Message bundles are simple properties files with a simple *parameter=value* structure. To display the *Welcome* on the homepage of the application, *Home.html*, depending on the web browser locale, you need a *messages_[countrycode].properties* file. For *myvideos* in German, this would be *messages_de.properties* in the */myvideos/context/WEB-INF* directory:

```
org.trails.welcome=➥
Willkommen zu Trails
```

Don't forget to replace the line *<h1>Welcome to Trails</h1>* with the line *<h1>Welcome to Trails</h1>* in *Home.html* to display a localized form of the welcome message.

Unfortunately, this approach does not work for the scaffolding: the homepage of the sample application still displays "List Movies". In his blog at [8], the developer promised a solution for this in the next version. Thus, it makes sense to add country-specific details to the scaffolding in anticipation of the change. And if you check out the CVS repository for the Trails source code, you can see that the packages are there.

An additional configuration file is required by Trails, and this file is not only for localization: */myvideos/context/WEB-INF/applicationContext.xml* integrates the Spring framework with Trails. It not only references the beans required for Trails and Hibernate, it also decorates the beans with various properties; and

Listing 3: Movie (2)

```
01 package de.wartala.myvideos;
02
03 import java.util.HashSet;
04 import java.util.Set;
05
06 import javax.persistence.
   CascadeType;
07 import javax.persistence.
   Entity;
08 import javax.persistence.
   GeneratorType;
09 import javax.persistence.Id;
10 import javax.persistence.
   JoinColumn;
11 import javax.persistence.
   OneToMany;
12
13 import org.apache.commons.
   lang.builder.EqualsBuilder;
14 import org.hibernate.
   validator.NotNull;
15 import org.trails.descriptor.
   annotation.Collection;
16 import org.trails.descriptor.
   annotation.PropertyDescriptor;
17 import org.trails.validation.
   ValidateUniqueness;
18
19 @Entity
20 @ValidateUniqueness(property="
   title")
21 public class Movie {
22     private Integer id;
23     private String title;
24     private Integer year;
25     private Set<Actor> actors
   = new HashSet<Actor>();
26
27     @PropertyDescriptor(index=
   0)
28     @Id(generate=GeneratorType
   .AUTO)
29     public Integer getId() {
30         return this.id;
31     }
32
33     public void setId(Integer
   id) {
34         this.id = id;
35     }
36
37     @PropertyDescriptor(index=
   1)
38     @NotNull
39     public String getTitle() {
40         return title;
41     }
42
43     public void setTitle(String
   title) {
44         this.title = title;
45     }
46
47     @PropertyDescriptor(index=
   2)
48     public Integer getYear() {
49         return year;
50     }
51
52     public void setYear(Integer
   year) {
53         this.year = year;
54     }
55
56     @PropertyDescriptor(index=
   3)
57     @OneToMany(cascade=Cascade
   Type.ALL)
58     @JoinColumn(name="movieId"
   )
59     @Collection(child=true)
60     public Set<Actor>
   getActors()
61     {
62         return actors;
63     }
64
65     public void
   setActors(Set<Actor> actors)
66     {
67         this.actors = actors;
68     }
69
70     public boolean
   equals(Object obj) {
71         return EqualsBuilder.
   reflectionEquals(this, obj);
72     }
73
74     public String toString() {
75         return this.getTitle();
76     }
77 }
```

this is also the right place to integrate message bundles like the ones shown in Listing 4.

The values of this parameter can be referenced in different ways within the Tapestry pages. As validation takes place within the domain object, it is also possible to append the output to possible field validators:

```
@NotNull(message="
{error.emptyMessage}")
@Pattern(regex=
"[A-z|\\s]+",message=
"{error.letterOrSpace}")
```

To keep the configuration of Trails as simple as possible, the in-memory version of HSQL [9] is integrated as the default database. This database is also used by OpenOffice. For all other databases, developers need to edit the *hibernate.properties* file in the *root directory/myvideos/context/WEB-INF* directory. But be careful: Hibernate does not support every database. Check out [10] and [11] to find out which databases are supported.

Ajax Support

Within the application scaffolding, a file titled *[applicationname].application* configures the Tapestry environment. The file references the Ajax library, Tacos [12]. One major advantage that Ajax offers is partial rendering of page components. This is useful for large pages with many input and output components. When a user is waiting for a web application, it makes a big difference whether the browser has to redraw 20 input fields in a web form or just one field. Tacos and thus Trails, has a simple approach for how to handle this. Unfortunately, the web developer not only needs to modify the HTML page, but also to implement a model class to do this in Tapestry. But again, help is on its way: the developer promises a simpler approach with the next release. If you can't wait for the next release, check out the Ajax example at [2].

Pretty Good

Just like it is impossible to say Rails without saying Ruby, it is impossible to say Trails without saying Tapestry if you intend to develop large-scale projects. Rails owes most of its power to the performance of the dynamic Ruby programming language. Java does not support dynamic attributes and methods (at this time of writing). In fact Rails has many goodies that the current Trails release lacks – email support, for example, although email should be easy to implement using Javamail. A convenient web services interface is also missing, although the Apache Webservice framework, Axis [13], should take care of this. On the other hand, Trails has the ability to link to existing databases via Hibernate, in contrast to Rails' O/R mapper ActiveRecord.

Admittedly, there is still a lot of work to do, but Trails version 0.8 is still a couple of steps away from its 1.0 release. If you take the trouble of checking out the current Trails version from the CVS repository, you will find references to features in the next version of Trails. Besides the I18N support referred to earlier, the Trails framework can look forward to a new package based on the Acegi security framework for Spring [14]. This security package will allow developers to move security-based declarations to the application context definition, in typical Spring style. And if Chris Nelson then polishes up the documentation, and clever developers find a way of integrating Java Server Faces instead of Tapestry, Trails may become a popular alternative for creating modern J2EE web applications. ■

Listing 4: Message Bundle

```
01 <!-- Message source for this context, loaded from localized
   "messages_xx" files -->
02 <bean id="messageSource" class="org.springframework.context.
   support.ResourceBundleMessageSource">
03     <property name="basename">
04         <value>messages</value>
05     </property>
06 </bean>
07
08 <bean id="trailsMessageSource" class="org.trails.i18n.DefaultTrails
   ResourceBundleMessageSource">
09     <property name="messageSource"><ref local="messageSource"/></
   property>
10 </bean>
```

INFO

[1] "On the Right Track: Web Applications with Ruby and Rails," by Armin Röhrl, Stefan Schmiedl, Linux Magazine 1/05, p. 62.

[2] Trails at *http://trails.dev.java.net*

[3] Jakarta Tapestry: *http://jakarta.apache.org/tapestry*

[4] AspectJ: *http://www.eclipse.org/aspectj*

[5] Spring Framework: *http://www.springframework.org*

[6] EJB 3.0 Annotation (JSR-220) at *http://www.jcp.org/aboutJava/communityprocess/edr/jsr220*

[7] Hibernate Annotations at *http://www.hibernate.org/hib_docs/annotations/reference/en/html*

[8] Chris Nelson's Weblog: *http://jroller.com/page/ccnelson/Weblog?catname=/Trails*

[9] HSQL: *http://hsqldb.org*

[10] Databases officially supported by Hibernate: *http://www.hibernate.org/260.html*

[11] Databases unofficially supported by Hibernate: *http://www.hibernate.org/80.html*

[12] Tacos: *http://tacos.sourceforge.net*

[13] Apache Axis: *http://ws.apache.org/axis*

[14] Acegi Security Framework for Spring: *http://acegisecurity.org*

THE AUTHOR

Ramon Wartala is a Software Engineer with AOL Germany, where he works in the Development & Architecture department. Whenever he takes time out from the battle against the Internet, Ramon likes to spend time with his wife.