

EMF State Machines

Markus Voelter, voelter@acm.org, www.voelter.de

PRIMARY SPONSORS



SECONDARY SPONSOR



Table of Contents

INTRODUCTION.....	3
INSTALLATION.....	3
METAMODEL.....	3
EXAMPLE STATEMACHINE.....	4
RUNNING THE EXAMPLE.....	5
THE GENERATOR.....	5
WORKFLOW.....	5
CONSTRAINTS	5
TEMPLATES.....	6
RECIPE CREATION.....	6

Introduction

This example shows how to implement a state machine generator using EMF and openArchitectureWare. Note that the implementation of the state machine in Java is probably the slowest and clumsiest way to implement a state machine. The focus was not on optimizing the performance of the state machine implementation.

This tutorial does not explain too much – it's rather a guide through the example code. We expect that you know how to work with openArchitectureWare and EMF. If that's not the case, you should read and play with the *emfHelloWorld* example first (the tutorial entitled *Generating Code from EMF Models*).

Installation

In the *emf examples* package, you can find the following three projects

- *oaw4.demo.emf.statemachine*: contains the metamodel
- *oaw4.demo.emf.statemachine.generator*: contains the code generator
- *oaw4.demo.emf.statemachine.example*: contains an example state machine as well as a manually written unit test

You have to install all the projects into your workspace. You also have to (as with all other examples) set the following three Eclipse classpath variables:

Variable	... points to
OAW_CORE	oaw4/oaw-core-4.x.x
OAW_LIB	oaw4/oaw-core-4.x.x/lib
OAW_RECIPE_CORE	oaw4/oaw-recipe-core-4.x.x

Metamodel

The metamodel looks more or less as you'd expect from a state machine metamodel. The following is the representation of the metamodel in Emfatic syntax. You can find it in the *oaw4.demo.emf.statemachine/model* package.

```
@namespace(uri="http://oaw/statemachine",
prefix="statemachine")
package statemachine;
```

```
abstract class Named {
    attr String name;
}

class State extends AbstractState {
    val Action entryAction;
    val Action exitAction;
}

class StartState extends AbstractState {
}

class StopState extends AbstractState {
}

class Transition extends Named {
    ref AbstractState[1] target;
    val Action action;
    ref Event event;
}

class Action extends Named {
}

class Event extends Named {
}

class CompositeEvent extends Event {
    val Event[*] children;
}

class StateMachine extends Named {
    val AbstractState[*] states;
    val Event[*] events;
}

abstract class AbstractState extends Named {
    val Transition[*] transition;
}
```

From the .ecore file, you have to generate the implementation classes – as usual with EMF.

Example StateMachine

In the *oaw4.demo.emf.statemachine.example/src* folder you can find an *example.statemachine* file that contains a simple example state machine. You can view it as an EMF tree view after generating the EMF editors.

To generate code from it, run the *example.oaw* workflow file right next to it. It looks as follows:

```
<workflow>
  <cartridge file="workflow.oaw">
    <modelFile value="example.statemachine"/>
    <srcGenPath value="src-gen"/>
    <appProject
value="oaw4.demo.emf.statemachine.example"/>
    <srcPath value="man-src"/>
  </cartridge>
</workflow>
```

As you can see it only defines a number of parameters and calls another workflow file – the one in the generator project. We'll take a look at it below.

Running the example

... is achieved by running the *example.oaw* file. It creates an implementation of the state machine in the *src-gen* folder in the example project. Take a look at the file to understand the implementation of the state machine.

In the *man-src* folder, there's a manually written subclass that implements the actions referenced from the state machine. There's also a unit test that you can run to verify that it works. It also shows you how to use the generated state machine.

The generator

Workflow

The workflow file in *oaw4.demo.emf.statemachine.generator/src* has four steps:

- first it reads the model from the XMI file
- then it verifies a number of constraints
- then it generates the code
- and finally it creates a recipes file

Constraints

A number of constraints are defined. Take a look at their definition in *structure.chk* to learn about the constraints check language.

Templates

In the *src/templates* folder you can find the code generation templates.

Recipe Creation

In *src/recipe* you there's an *SMRecipeCreator* workflow component that creates recipes for the manual implementation of the state machine. Recipe Creation

Acknowledgements

Thanks to the folks at Rohde & Schwarz Bick Mobilfunk for letting us use this example.

About our Sponsors

itemis GmbH & Co. KG is an independent IT service company with an emphasis on consulting, coaching, and software development. Every single itemis expert provides many years of project experience and widespread knowledge about all object oriented and component based software development issues - especially in the field of model driven software development.

b+m is the founder of the openArchitectureWare project. The software was originally developed within the scope of many successful projects. b+m opened the software to the community in late 2003. All of the paradigms of Model-Driven Software Development including Product Line Engineering and not only the generator framework have become a key concept for product and customer specific development at b+m. b+m customers can make use of long time experience and substantial know-how in that field. Located at the company headquarters in Melsdorf/Kiel and at its subsidiaries in Berlin, Cottbus, Hamburg, Hanover and Kiel the b+m staff of 205 provides practical solutions for customized business applications, business process optimization and comprehensive architecture, project and quality management.

oose Innovative Informatik GmbH offers coaching, consulting and training in all themes about software engineering. The main focus of their activities are software architecture, requirements engineering and project-management. oose have first-hand information and experience, because our staff take actively part with others in actual trends, standards and innovations. Our staff support this and pass their know-how regularly on by writing and publishing books or being speaker at conferences, etc. Within the OMG oose collaborate actively on the specifications of the UML and also the SysML.

MID Enterprise Software Solutions GmbH is a leading supplier of optimized tool environments for standardsbased and model-centric software development as well as business process modeling. This includes professional tool consulting and tool components to build a complete tool environment using the best techniques and tool modules available - Architectural and Operational Excellence. With innovatorAOX, MID provides a holistic standard tool environment for object- and function-oriented software development as well as business process and data modeling to help its customers establish highly efficient processes and tool environments for software production, The unique and seamless integration of business process modeling into the development process ensures an unprecedented level of convergence of business requirements and implemented IT systems. Project members from all departments speak the same language and all requirements are clearly described.