Proceedings
of the 4th International Modelica Conference,
Hamburg, March 7-8, 2005,
Gerhard Schmitz (editor)

S.E. Pohl, J. Ungethüm
DLR Stuttgart, Germany

A Simulation Management Enviroment for Dymola

pp. 173-176

Paper presented at the 4th International Modelica Conference, March 7-8, 2005,
Hamburg University of Technology, Hamburg-Harburg, Germany,
organized by The Modelica Association and the Department of Thermodynamics, Hamburg University of Technology

All papers of this conference can be downloaded from
http://www.Modelica.org/events/Conference2005/

A Simulation Management Environment for Dymola

Sven-Erik Pohl* Jörg Ungethüm†

German Aerospace Center (DLR), Institute of Vehicle Concepts
Pfaffenwaldring 38-40, 70569 Stuttgart

Abstract

Building Modelica libraries is a steady process of adding and refining models. There is rarely a final version of a library. This leads to the fact that simulation results are difficult to reproduce due to changes in sub-models. However, reproducible simulation results have to be provided for solid project work and scientific research. Using Matlab as a platform a simulation management environment, called SiME, was developed. This environment includes general simulation handling, as well as tools for pre- and postprocessing. A HTML-based simulation history containing parameters and results is included.
Keywords: Modelica tools, simulation management, Matlab, CVS

1 Introduction

Developing Modelica libraries while at the same time maintaining several complex models can make backward compatibility difficult. Moreover, keeping specific library versions is especially important if simulation results were published in research reports or scientific publications. A common practice in software engineering, a version control system appears to be an adequate solution to manage the code evolution. However, if several libraries are involved in the simulation keeping track of the files becomes tedious. Additionally, linking the versions with the results has to be done manually.

2 Objectives

A simulation management environment was outlined to overcome this shortages. The central idea is to automate and standardize the versioning process. It is necessary to apply versioning not only to the Modelica libraries but also to any auxiliary files like pre- and postprocessing scripts, which are necessary to run the simulation. The ability for recovering the model and rerun the simulation is sensible against the completeness and version-correctness of these auxiliary files. Multiple simulation projects must be supported, even if libraries or scripts are used and developed concurrently. Another objective is a clear and informative simulation report which includes version information as well as simulation results.

3 Design

The simulation management environment tends to become a very complex system, as various different components are needed to reproduce simulation results. Besides the core model code, parameter lists, measured data, experiment scripts, documentation and other auxiliary files must be stored and recreated. Format, size or number of these auxiliary files is not known in advance. Nonetheless, reliability is major concern for the simulation management environment. However, a substantial ambition in the design process was a straight and simple realization. Therefore, the Concurrent Versions System (CVS) [1] was employed as base layer. CVS uses a client/server architecture, which makes it easy to install and maintain. SiME itself consists only of client side scripts and does not require any additional server software. Therefore SiME can be used with any CVS server. Due to the use of a standard version control system, any file within SiME is accessible using standard tools. Furthermore, concurrent access using SiME and standard CVS clients is seamlessly possible. The following code fragment shows the Matlab call of the CVS client.

```
syscmd=[cvsbin_name,' ',cvsopt_str,' ',...
       cvscmd, ' ',cmdopt_str,' '];
for i = 1:length(cmdargv)
  cmdarg_str = deblank(cmdargv{i});
```

*sven.pohl@dlr.de
†joerg.ungethuem@dlr.de

release	comment	date	time	path	mainscript	results
fklg0001	a first test run	2004-03-16	13:13:56	.	fklg_dymola.m	fklg0001.html
fklg0002	Comparison of four gas springs to investigate impact of heat transfer and blowby	2004-03-30	10:46:59	.	fklg_dymola.m	fklg0002.html
fklg0003	Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring	2004-05-26	14:31:41	.	fklg_dymola.m	fklg0003.html
fklg0005	Correct IdealGasFormulation: Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring	2004-05-28	13:19:14	.	fklg_dymola.m	fklg0005.html
fklg0006	Correct IdealGasFormulation: Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring. (Using new release of plotmaster)	2004-06-01	13:05:22	.	fklg_dymola.m	fklg0006.html

Figure 1: Automatically created simulation overview html-page

```
  cmdlen=length(syscmd)+length(cmdarg_str);
  [rcc,mesg]=system([syscmd,cmdarg_str]);
end
```

The CVS command part is defined by `syscmd` while the files to processed are bundled in `cmdargv`.

The core of SiME consists of a few scripts written in Matlab. Dymola's simulation results are easily accessible within Matlab. Matlab's numerical capabilities are outstanding and within SiME all of its features, e.g. the data visualization tools are applicable. However, Matlab is not optimal for string processing and system command execution. Dedicated scripting languages like python and Perl are much more comfortable in this context.

4 Features

The basic concept of SiME is to organize arbitrary simulation tasks in projects. Each project consists of a history of completely reproducible simulation runs. For example, a project named "hybridcar" is a Modelica library development of a hybrid electric vehicle. The developer uses SiME to protocol the developing process and to document the evolution steps. It's not only possible to directly compare the results but also to retrieve the complete simulation code, to make changes if necessary and to re-run the simulation.

The Simulation Management Environment splits each simulation process into four steps. The initialization part sets up the simulation run and calls the CVS routines. Preprocessing, simulation and postprocessing mainly contain code to handle the simulation application (e.g. Dymola) and its results. In a possible fifth step the complete simulation run can be repeated simply using the simulation ID.

4.1 Initialization

During initialization a unique simulation ID is generated and a complete list of all files which are relevant for the simulation is built. Matlab script dependencies are collected automatically. However, Modelica and auxiliary files must be added manually. SiME forces any file on the list under version control if this was not done before:

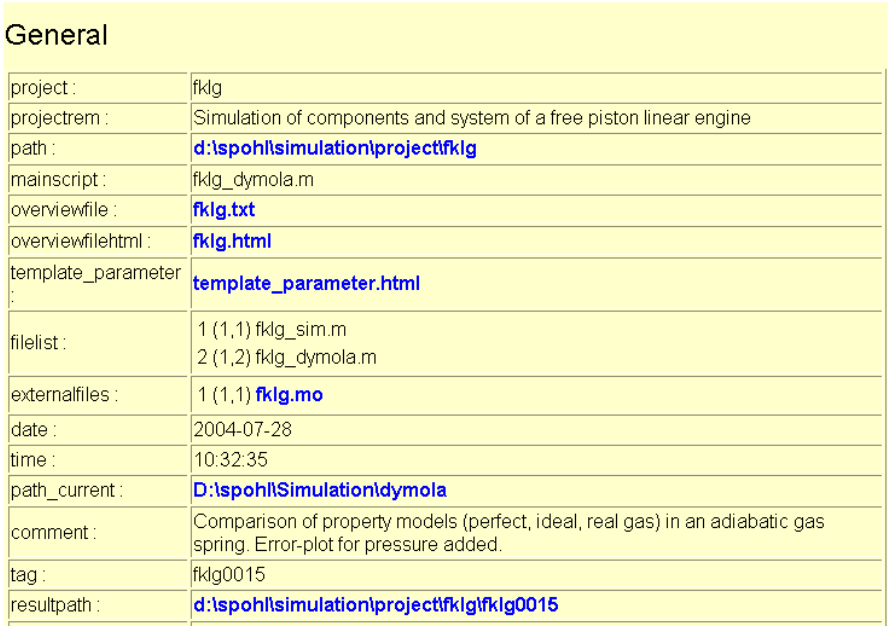
```
for element=1:counter
[err,errmsg]= fkcvsadd('',...
cmdopt,notinrepository(counter));
end;
```

Afterwards the files are checked in and tagged using the simulation ID. In that way all files involved in the simulation process are marked with the simulation ID and can be retrieved securely. This code fragment illustrates the process of committing and tagging:

```
for i=1:filenum
  % commit files
  cmdopt.m = ['automated commit'];
  [err.commit(i),errmsg.commit{i,1}]=...
  fkcvscommit('',cmdopt, ...
  remainder(startpos(i):endpos(i)));
  % tag files
  [err.tag(i),errmsg.tag{i,1}]=...
  fkcvstag(tag,'','', ...
  remainder(startpos(i):endpos(i)));
end;
```

4.2 Preprocessing

The initialization routine calls the main simulation script. From this Matlab script the external simulation



General

project :	fklg
projectrem :	Simulation of components and system of a free piston linear engine
path :	d:\spohl\simulation\project\fklg
mainscript :	fklg_dymola.m
overviewfile :	fklg.txt
overviewfilehtml :	fklg.html
template_parameter :	template_parameter.html
filelist :	1 (1,1) fklg_sim.m 2 (1,2) fklg_dymola.m
externalfiles :	1 (1,1) fklg.mo
date :	2004-07-28
time :	10:32:35
path_current :	D:\spohl\Simulation\dymola
comment :	Comparison of property models (perfect, ideal, real gas) in an adiabatic gas spring. Error-plot for pressure added.
tag :	fklg0015
resultpath :	d:\spohl\simulation\project\fklg\fklg0015

Figure 2: Automatically created simulation report html-page

application is started. Generally, any application using the Dynamic Data Exchange (DDE) interface can be called remotely. In this case Dymola is used. In the preprocessing part the Dymola model and the model parameters are defined. For parameter studies an array of values for each parameter can be provided. A parameter matrix is then built from the parameter arrays.

4.3 Simulation

Dynasim provides Matlab functions to start Dymola and execute commands via DDE interface. These functions are used to establish an interaction between Matlab and Dymola. The self explanatory code is shown below:

```
% Set up experiment
ex=dymoexperiment;  % default values ex
ex.StopTime = 0.1;  % set StopTime

% Start Dymola
 res.start = dymostart(sim.dymolabinpath);
% Load Package
 res.load = dymoload(sim.package);
% Check Model
 res.check = dymocheck(sim.model);
% Translate Model
 res.translate = dymotranslate(sim.model);
for num = 1:sim.parmatrixsize
  % Set parameter(s)
    dymosetpar(sim,num);
  % Simulate Model
    res.simulate = dymosimulate(...
    sim.model,ex,sim.modelname);
end;
% Close Dymola
 dymoexit;
```

4.4 Postprocessing

Subsequent to the simulation process arbitrary Matlab scripts can be run to further process the results, e.g. to generate plots.

A standardized protocol in HTML is generated including the history of simulation tasks. An example of this simulation overview is shown in Figure 1. For every simulation task a link to a HTML report is given. The HTML report (Figure 2) includes in detail all the parameters used within initialization, preprocessing, simulation process and hyperlinks to the saved plots. Additionally, a summary of all error messages and comments occurred during the runtime is given.

4.5 Re-Run Simulations

To retrieve the complete set of simulation files only the simulation ID is needed. A MATLAB function will retrieve the files from the repository. The files are now ready for manipulation and a new simulation run can be started.

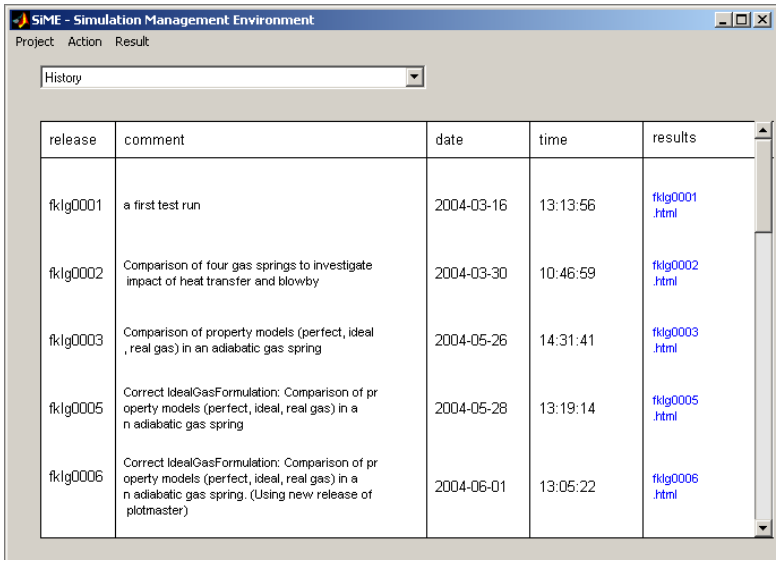


Figure 3: The SiME graphical user interface

5 Graphical User Interface

To coordinate the projects and their history a graphical user interface is added. Figure 3 shows the main interface window. Projects can be loaded, created and managed. General information, e.g. directories and project details, can be edited. The simulation history of projects can be browsed and simulation results displayed. New simulation runs can be tested or launched. Dymola result files can be browsed using the Matlab GUI provided by Dynasim.

6 Limitations

SiME inherits the limitations of the CVS system. E.g. the handling of binary files like pictures is not optimal and reordering of directory structures is difficult and error prone.
Most of the CVS limitations are overcome by its successor, the Subversion [2] system. Subversion reached release 1.0 in March 2004. Currently little practical experiences with Subversion are present. However, as Subversion is downwards compatible to CVS, switching to Subversion should be possible.
Absolute directory path references might be included in the model code, like script references in Dymola's annotations. These path references become staled links in the recreated files which cannot be fixed easily.
The Matlab–Dymola communication uses Dymola as DDE server which is available on MS Windows only. This prevents the SiME client currently from working on any other operation system.
Some additional Dymola scripting language commands would be desirable, e.g. retrieving a list of all Modelica files which are referenced by a model or generating a screen-shot of the current diagram window.

7 Conclusions

A simulation management environment (SiME) was developed to provide easy and efficient access to earlier simulation runs. SiME ensures the reproducibility of simulation results, even if the models involved are still in development. This facilitates the documentation, avoids redundant work and is an important contribution for quality assurance.
SiME uses Matlab as scripting language, since Matlab is used frequently already for pre- and postprocessing. As backbone server for SiME the Concurrent Versions System (CVS) was selected, since it is freely available and extremely reliable. The use of the models outside of the simulation management is not restricted, so that the normal, CVS supported model development is not disturbed.

References

[1] Version Management with CVS. Per Cederqvist. https://www.cvshome.org/docs/manual/

[2] Version Control with Subversion. Ben Collins-Sussmann, Brian W. Fitzpatrick, C. Michael Pilato. http://svnbook.red-bean.com/en/1.1/svnbook.pdf