MADES R1.0, USER MANUAL

Baresi Luciano
Politecnico di Milano
baresi@elet.polimi.it

Bonvini Marco
Politecnico di Milano
bonvini@elet.polimi.it

Ferretti Gianni
Politecnico di Milano
ferretti@elet.polimi.it

Leva Alberto
Politecnico di Milano
leva@elet.polimi.it

Rossi Matteo
Politecnico di Milano
rossi@elet.polimi.it

Sama Michele
PuzzleDev s.n.c.
m.sama@puzzledev.com

Abstract MADES is a co-simulation tool aiming to generate feasible execution traces for systems composed by a software component and an physical external device described in terms of its mechanical, electrical, and thermochemical properties which represent the environment in which the software component is running. The software component is simulated with a discrete time step representing the sampling time of sensors, while its environment is simulated with a continuous time representing the physical reaction of the system. The generated trace is deterministic for the environment, and non-deterministic for the software, since it may include human interactions. In this document we give a general overview of MADES and of its components and we explain in details how it can be used to generate co-simulated traces.

Keywords: MADES, ZOT, Modelica, trace generation, co-simulation

1. General Overview

MADES is an open source co-simulation tool aiming to generate realistic execution traces for software components driving external devices inside a physical environment. The goal of MADES is to co-simulate both the software and the environment simultaneously and consistently. This is done by means of a set of shared timed variables. The system declares a set of output variables which are used as input for the environment. Similarly the environment declares a set of output variable which are given in input to the system.

The co-simulation simulates the system with a discrete time, which represents the software response time. We call that delta time a simulation *step*. Each step represent the time between the current co-simulation time and the following time plus δ, where δ is the constant time duration of each steps. At each step of the co-simulation the environment is simulated with a continuous time starting from the current time t until $t + \delta$.

After each step all the shared timed variables are flagged with both the continuous time $t + \delta$ and the discrete time s represented by the number of steps. For instance at the third step of co-simulation all the produced variables are flagged with $t = 3 * \delta$ and $s = 3$.

The trace produced by MADES at the end of the co-simulation is the ordered set of all the shared variables from the beginning till the end of the co-simulation. Such trace is saved as in a file called *madesOutput.xml* placed in the same folder of the simulated model.

1.1 Simulators

MADES uses an open architecture which allows developers to use their own simulators during the co-simulation. At the moment the binding is static and simulators are specified at runtime, however in the future a plug-in based architecture will be implemented allowing dynamic selection of the various simulators.

The current version uses the temporal-logic solver ZOT [Pradella, 2009] and the sat solver z3 [de Moura and BjÃÿrner, 2008] to simulate the system and uses the Open Modelica Compiler (omc) for the environment [Fritzson et al., 2005].

The co-simulations requires in input a ZOT and a Modelica models which will be instrumented and executed by MADES. For more informations about the input files see Section 2. For information about how the model are instrumented see Section 3. For technical details in how to use a different simulator please see Section 5

1.2 Downloading And Running MADES

There are to ways to download MADES. Developers willing to extend the co-simulator can download the latest source of MADES from the repository on Google code [PuzzleDev s.n.c., 2011]. Users willing to simply run a co-simulation can download a re-distributable package of MADES from the download section of the Google code MADES website.

MADES is entirely written in java 1.6 and therefore it requires the JRE to be installed. During our development we used the OpenSDK 1.6 and that is the evironment that we recommend. In order for MADES to run correctly there are several dependencies which have to be satisfied. MADES is supposed to be executed in a linux-like environment. The suggested environment is Ubuntu 11.04. In addition MADES requires gcc, omc and the list interpreter sbcl to be installed. Note that the Open Modelica Compiler must be of a version equals or newer than 1.7.0 (r9303). In addition ZOT and z3 to be installed and available to the current user. For convenience we included the version of ZOT that we used during the development in the download section. For licenses issues we could not include z3, however it is also available for free download [Microsfot Research, 2011]

To facilitate the users we provide a bash install script which will automatically download and install all the required libraries. The install script is also available from the download page in the MADES Google code website. Once MADES is currently installed it can be execute in two ways, as a shell command or with a GUI.

1.3 Running MADES with the GUI

Users willing to use MADES stand-alone should use the gui version. To do so the can simply double click on the *Mades.jar* file and it should run. Alternatively the MADES' main form can be opened by running:

```
java -jar Mades.jar
```

The GUI mode is the default mode to use MADES. Please note that if the GUI is invoked from a shell it will display in output all the log messages. Users having problems while simulating a particular model may find this output information very useful.

Figure 1 shows the main window of the MADES GUI. To run a simulation the user must first select a *mades.xml* file then press the start button. It is possible to specify different simulation parameters by changing the values on the main window.

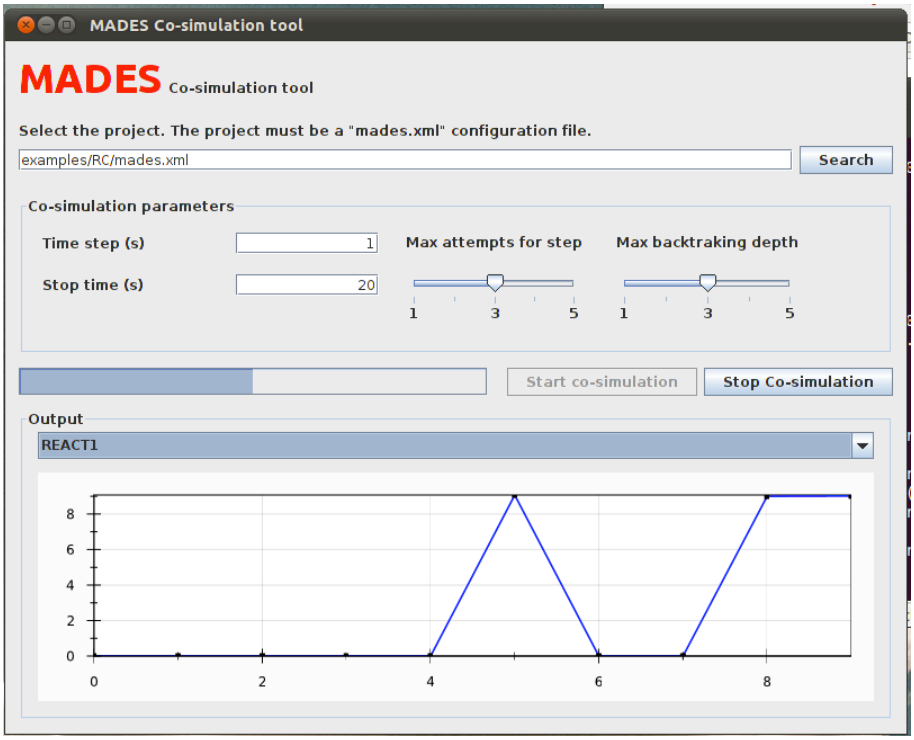

Figure 1. The MADES main GUI

At the bottom of the main window all the shared variables are plotted. Users may interactively zoom in and out as they please. They may also print or export the picture as PNG files.

1.4 Running MADES From The Command Linel

Users willing to run MADES as a shell command should run:

```
java -classpath ./:./Mades.jar mades.cosimulation.Cosimulator <model-name>
```

If we execute the previous command without any parameter MADES terminates with the following error:

```
Cosimulator
ERROR: filename string not specified.
Usage: Mades <filename>
             [-timeStep=1]
```

```
[-stopTime=20]
[-attemptsInStep=3]
[-backtrakingDepth=3]
```

In order to work from command line MADES must be invoked specifying a *mades.xml* file and all the required configuration parameters.

2. Input Models

To work properly MADES expects to find two set of files: the input models, which are specified by the user at runtime and a set of files used to instrument the given model as required by the co-simulator. Table 1 list the ones of which users should be aware of. All the other files are either generated during the compilation or created at runtime as results of the computation.

Table 1. Mades input file

File name	Description
mades.xml	The MADES configuration file
zot/run.zot	The runner for the ZOT model
zot/MODEL_system.zot	The system model for ZOT
zot/MODEL_constraints	The system constraints for ZOT
zot/MODEL_history.zot	The system constraints created by the co-simulator during the execution
modelica/sources/MODEL.mo	The source file of the environment model for Modelica
modelica/MODEL	The instrumented and compiled model for Modelica
modelica/MODEL_init.xml	The initial parameters for Modelica
modelica/MODEL_res.csv	The simulation results generated by Modelica at each step
madesOutput.xml	The trace file generated at the end of the cosimulation

The file *mades.xml* contains a description of how the system and the environment interacts. All the shared variables must be defined here. If necessary also private variables belonging to the system or to the environment can be declared here. For convenience we declared here also ZOT private variables.

In addition to variable definition the xml file contains *trigger groups* and *trigger* definitions. A trigger represent a variable being monitored. Monitored variables are associated to a threshold value and with a boolean signal. When the monitored variable has a value lower than the threshold their associated signals has value 0. Contrarily when the variable is higher than the threshold its associated signal has value 1.

Variables are monitored both to have trigger signals and to assure that they do not change their value more often than δ. Each time a monitored variable changes their value Modelica trace them and invalidates the last step of the simulation.

If a trigger group contains more than a variable not only the single variables must not cross their associated threshold more often than δ but also when one changes either the other change values simultaneously or the should not change sooner than δ.

3. Models Instrumentation

The MADES co-simulator instruments the given models with additional instructions required to ensure the correctness of the generated trace. The instrumentation depends on the simulation tools used.

The ZOT model is instrumented in order to keep trace of the previously simulated steps. The file *run.zot* is added to the system model and instrumented in order to run the given model with an history extracted from the trace generated so far. In addition whenever a simulated step is rejected or when the co-simulator backtrack a previously accepted step its configuration is added to the history as a configuration to be avoided.

The Modelica model is instrumented by adding trigger signals for the monitored variables and to invoke a print function every time a threshold is crossed. It is then recompiled before simulating the first step. By design the Modelica model is initialized with a set of initialization parameters located in the file *MODEL_init.xml*. At each simulation step this file is instrumented with values resulting from the previous simulation steps. Similarly Modelica produces a result file called *MODEL_res.csv* which contains all the simulated values. The co-simulator reads the output values at time $t + \delta$ and updates the values in the *MODEL_init.xml* file.

4. How the Co-simulation Works

The co-simulator starts from a given initial state and performs the simulation of both environment and system. At each step, starting from the configuration of the previous step, first the environment is simulated then the system is simulated using an intermediate configuration representing the system configuration at time t and the system variables at time $t + \delta$. The result produced by this co-simulation are correct under the assumption that all the shared variables keep their value for the whole step and that they change only at the end of each step.

Since the environmental variables simulated by Modelica are continuous and since they may change their value at any time the component wrapping Modelica must guarantee that they do not *change* too quickly, where changing means to cross a threshold line. This is done by means of signals and signal

groups. Internally the Modelica wrapper informs the co-simulator if one of the signals or one of the signal groups has changed sign within a time shorter than δ. When that happens the co-simulator rejects the simulation step and rolls back to the previous accepted environment simulation. Similarly Zot may fail to find a suitable solution if the given configuration is not satisfiable. Also in that case the the co-simulation rolls back.

Please note that since Modelica is deterministic repeating the same simulation twice will produce the same output. Once a not satisfying configuration has been reached the co-simulator needs to roll back not only Modelica but also the latest step simulated by Zot. In order to prevent Zot from re-simulating the same trace the co-simulator add the rejected step in the history of the Zot model as a configuration to avoid.

If after a certain number of attempts no valid configuration is generated the co-simulator rolls back to a previous step, then try again. This continues until the maximum backtrack limit is hit. If the limit is reached then the co-simulation fails.

5. Expanding MADES

MADES is structured as a set of pluggable components which can be replaced at compile time. Future versions of MADES will allow developers to select which simulator to use at runtime.

The co-simulator invokes the system by calling a component implementing the interface *SystemConnector*. Similarly the environment is accessed by invoking a class implementing *EnvironmentConnector*. Internally those components wraps a simulator and invoke it as needed. The configuration resulting from each simulation step is stored using the Memento design pattern. Instances of the class *SystemMemento* store the system configuration while instances of *EnvironmentMemento* stores the environment state.

Developers willing to connect a third party simulator to modelica should implement a wrapper implementing the opportune connector interface and should return an instance of the right memento at each invocation.

References

[de Moura and BjÃÿrner, 2008] de Moura, Leonardo and BjÃÿrner, Nikolaj (2008). Z3: An efficient smt solver. In Ramakrishnan, C. and Rehof, Jakob, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer Berlin / Heidelberg.

[Fritzson et al., 2005] Fritzson, Peter, Aronsson, Peter, Lundvall, HÃěkan, NystrÃűm, Kaj, Pop, Adrian, Saldamli, Levon, and Broman, David (2005). The openmodelica modeling, simulation, and development environment.

[Microsfot Research, 2011] Microsfot Research (2011). Z3 sat solver. http://research.microsoft.com/en-us/um/redmond/projects/z3/download.html.

[Pradella, 2009] Pradella, Matteo (2009). A user's guide to zot. *CoRR*, abs/0912.5014.

[PuzzleDev s.n.c., 2011] PuzzleDev s.n.c. (2011). Mades svn repository. http://code.google.com/p/mades/.