Project	ICESTARS
Project Number	FP7/2008/ICT/214911
Work Package	WP3
Tasks	T3.2 and T3.3
Deliverable	D3.3 - version 2.0

title	Software modules for RF boundary conditions and transient simulations
authors	Sascha Baumanns, Michael Matthes Caren Tischendorf, Monica Selva Soto Wim Schoenmaker, Peter Meuris , Gabor Bella Bart De Smedt, Wim Verhaegen
Affiliations	University of Cologne MAGWEL NV
date	May 20, 2010

ICESTARS FP7 / 2008 ICT 214911 D3.3

Contents

1 Abstract

The purpose of this document is to provide supporting material to the deliverable D3.3, which is a *software* deliverable. As such it should provide evidence for the realization of the software modules that were developed for transient simulations and the set up of mixed-mode simulations (coupled field - circuit simulations).

2 Introduction

The software modules that were developed in WP3 are focusing on three major themes. The first part of this document deals the DAE solver DAEn which was originally implemented in MATLAB by Michael Hanke, for background information see [1].
The second part deals with the software modules that were developed for coupling the field solver and the circuit solver. Finally, the third part deals with the software realization of the transient field solver.

3 DAEn software

The DAEN code was slightly modified and transferred to Python. DAEn is a BDF solver using a combination of error estimation and order selection strategies. The solver DAEn is restricted

to DAEs of the standard form

$$M(t)x' + f(x,t) = 0$$

with a mass matrix M. M can be constant or time dependent, but should not depend on x. For stability reasons, this form is not recommended for solving circuits with time-dependent or non-linear dynamic components. Therefore, a Python implementation of a BDF solver for solving DAEs of the form

$$f(\frac{\mathrm{d}}{\mathrm{d}t}q(x,t), x, t) = 0$$

is under development. Despite of this restriction, the solver DAEn provides all functionalities needed for a transient EM simulation and a coupled transient field-circuit simulation.

3.1 Code: Basic design

We present briefly the basic files of the numcgnpy-package. For DAEn the following files are needed:

csparse.py	This file contains the main sparse matrix class Csparse.
exampledaes.py	All the example DAEs are given here as hard defined functions. They are organized in classes, e.g. class Eta_dae.
func.py	This file contains the classes Func and MFunc.
funcDAE_standard.py	In the class FuncDAE_standard the interface of a DAE with a DAE solver is specified. This is the main interface class for all DAEs of the form $Mx' + f = 0$.
daen.py	This file contains the DAEn class which represents the DAEn solver.
test_daen.py	This is the script which puts all the code pieces together, calls the objects and functions in the right order and plots the solutions. This is not a real test by now. It is just for experimenting purposes.

3.2 Step-by-step: Calling DAEn

For using the solver we have to do the following steps (have a look into test_daen.py):

1. As mentioned above DAEn tries to solve a DAE in standard from with a mass matrix M. So we need a DAE first. Therefore we have to define the matrix $M(x,t)$ and the function $f(x,t)$ first. If we know the Jacobian (w.r.t. x) of f we can define it as well. For a set of examples have a look into exampledaes.py. Make sure that the function representing M returns a Csparse matrix and f an array.

2. We then need to define M and f as MFunc resp. Func objects:

   ```
   fun=Func(n,m,f,fjacx)
   mass=MFunc(n,m,r)
   ```

   If we do not know the Jacobian of f it is computed numerically. It is the main idea of the Func objects to adjoin the Jacobian to the function itself. n, m, r are dimensions and it

can be specified as well if the matrices are constant. For further details have a look into func.py.

3. The next step is to define the dae itself via the FuncDAE_standard interface:

   ```
   dae=FuncDAE_standard(mass,fun,x0,interval)
   ```

   Now the DAE is characterized as an initial value problem on an interval. Here the starting value $x0$ must be an array and interval is a tuple containing the interval boundaries, e.g.:

   ```
   interval=(0.,1e-3)
   ```

4. Now we have to create the solver object. Therefore we define some options (a dictionary). If we do not do this the default options are taken. This might look like

   ```
   options={'abstol':1e-4,
            'reltol':1e-4,
            'estrat':2,
            'ostrat':'Std',
            'scaling': 'on',
            'report': 'none'}
   daesolver=DAEn(dae,options)
   ```

   A complete listing of the options can be found in daen.py in the DAEn class. The options can be changed as well after the the solver object is created via the set_opts function in DAEn.

5. Now we can start the solver by calling

   ```
   daesolver.run()
   ```

   This is the main method of the solver which computes a solution or fails.

6. If the computation was successful we can get the solution with:

   ```
   tout,xout = daesolver.get_solution()
   ```

   This can be plotted now.

4 Coupled field-circuit simulation

The underlying idea is illustrated in Fig. 1. The parser module `pyparse.py` generates the DAE formulation. An interface is constructed that realizes the coupling as is shown in Fig. 2. The parser module `pyparse.py` generates the DAE. formulation.

$$E * \frac{\mathrm{d}}{\mathrm{d}t} d(Y(t), t) + b(Y(t), t) + c(Y_{ext}, t) = 0 \quad (1)$$

$$g(X(t), Y(t), \frac{\mathrm{d}}{\mathrm{d}t} X(t), t) = 0 \quad (2)$$

The DAEs are the circuit equations from a SPICE net list with Y being the network variables (node potentials e, currents i_L through inductances, i_V through voltage sources and i_M through EM elements).

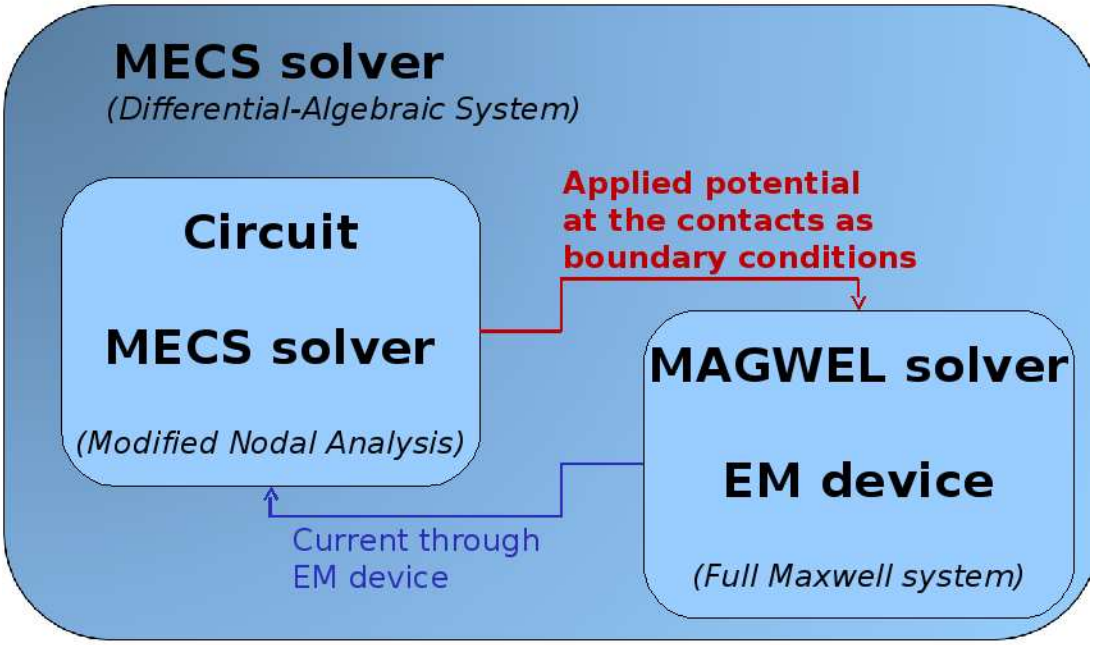

Figure 1: Flow schematics of the circuit-field coupling

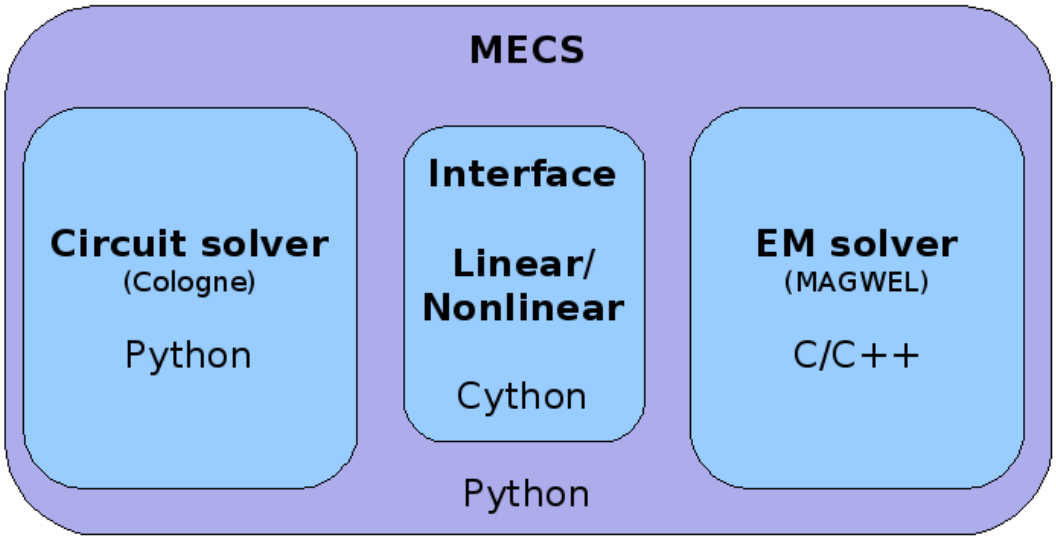

Figure 2: Technical realization of the circuit-field coupling

The transient MAGWEL solver provides the space discretized field description in the form (see deliverable D3.2)

$$M * \frac{\mathrm{d}}{\mathrm{d}t} X(t) + H(X(t), t) + F * Y(t) = 0 \quad (3)$$

$$G(X(t), Y(t)) = 0 \quad (4)$$

with X being the field variables (scalar potentials V, vector potentials A and $\Pi = \frac{\mathrm{d}}{\mathrm{d}t} A$).

For the coupled field-circuit simulation, the composed system (1)-(2) and (3)-(4) represents the differential-algebraic system to be solved by the DAE solver.
The communication bridge between C/C++ and Python, Cython, has the following characteristics:

- Data exchange via pointers to vectors and matrices no data copying!
- EM solver functionality is available via a shared library
- Calling order:
  - MECS starts EM solver (initialization, memory allocation, etc.)
  - MECS calls EM solver in every time getting the right hand side of the Maxwell system
  - After time integration MECS stops the EM solver (deallocation of memory, etc.)

The coupled system is solved with a DAE solver (BDF) as time integrator using adaptive step size control and order selection strategies.

5 Known issues, things to investigate

5.1 Coupling

The coupled system formulation (1)-(4) is implemented for a few examples. The general interface implementation between the field and the circuit description as FuncDAE interface is under development.

5.2 Parser

The parser module has to be extended to handle subcircuits and BSIM transistor models. So far, it can be used for linear circuits including very simple transistor models.

5.3 DAE solver

- DAEn is only recommended for circuits with constant dynamic network elements. Circuits with nonlinear and time dependent dynamic network elements can be solved by the DAE solver module BDFDAE. The implementation in Python is almost ready and has to be tested.
- DAEn can just be called for a predefined time span. After computation it returns the vector of time steps *tout* and the solution data *xout*. Of course, more information can be returned if needed, but it is not implemented so far.

5.4 Differences: Python vs. MATLAB code

There are a few things which are differently implemented:

- In the code many linear systems have to be solved. They rely on the LU-composition of the Matrix J (for the exact computations see in the code). In Python this is a Csparse matrix. Then the systems look like
$$Jx = b$$
where b is different for different systems. The LU-decomposition in Python is different from the one in MATLAB. At least technically. In Python first a LU object is created which represents the LU-decomposition:

```
lu_J=J.lu()
x=lu_J.solve_lu(b)
```

In the MATLAB version we get the matrices L, U, P, Q out of the LU command such that
$$PJQ = LU.$$
There is the possibility to choose a heuristic LU via the given parameters. By doing this Q is guessed. This saves 25 per cent of computation time. This feature is not implemented in the python code. It has to be checked if it is really necessary.

- We observed a slight difference in accuracy between Python and MATLAB.

- In the MATLAB code it has to be specified which type of mass matrix M or Jacobian of f we have. This is no longer necessary in the Python version. This can be looked up in the FuncDAE_standard interface.

5.5 Still to be checked

There are some things left which have to be done.

- The tested examples (see below) have dimension at most 15. Still the Python solver is not much faster than the MATLAB version or even slower. This has to be rechecked when we have larger examples and the sparse structures get more important. Furthermore the pure Python code needs to transferred to a Cython version making it a lot faster.

- All values are represented in the float64 format while other parameters like the step size are in normal float format. Here it needs to be checked if a conversion to double (=float64) should be done. Anyway if we introduce the Cython version this will be done.

- The error handling and the process information can be improved. The only error which is really handled is if the stepsize is getting too small. Then the run-method stops and the solutions which are computed so far can be obtained.

- If the jacobian of f is not not given it is computed numerically. This is done by the function numjac in the Func class. The exact Jacobian was always given in the examples below. This function needs to be optimized.

6 Using the MAGWEL solver for transient solver input

6.1 Matrix generation instructions

There are two groups of matrices that are generated using the MAGWEL field solver. The first set deals with generating the Newton-Raphson matrices in a Taylor expansion in the frequency (or time-derivative) variable. In order to produce these matrices it is required to modify the file "log4cpp_solvem.conf" as follows:

```
#
# log4cpp configuration
#

priority root INFO

# linear solver related
priority ComplexSparseSystem DEBUG
priority SparseSystem DEBUG

# ICESTARS related
priority hidden.features.dumpMatrixVFA DEBUG
priority hidden.features.ReducedOrderModeling DEBUG
priority hidden.features.MatrixTaylorExpansion DEBUG
priority  hidden.features.scaled_MTEXorROM_Dump DEBUG

appender root pattern console %p [%c] %m%n
```

The run command for generating the matrices that is provided by the command using the '-l' option.

```
solvEM -l log4cpp_solvem.cpp inputfile.xml
```

The availability of these modules was demonstrated in deliverable D3.2 where the example for the 'prototype' inductor was described. The purpose of that example was partly to test the correctness of the matrix generation. In the next subsections, we will now discuss the separate lines in the file.

6.2 priority hidden.features.dumpMatrixVFA DEBUG

Each line starts with a 'priority' declaration. This string is tested by the log4cpp category "getInstance" and if successful, a category pointer variable can be associated to it.

```
category* dumpMatrixVFA =
               &Category::getInstance("hidden.features.dumpMatrixVFA");
```

The third keyword requires a comparison. For that purpose, the pointer variables is dereferenced to a method 'isDebugEnabled' and the return value is a Boolean. The purpose of this dump is mainly interesting for debugging purposes and for use in the frequency regime.

```
bool test = dumpMatrixVFA->isDebugEnabled()
```

With the Boolean variable being 'true' the dumping of the system matrix is activated.

6.3 priority hidden.features.ReducedOrderModeling DEBUG

Just as above, the dumps of the matrices , B, C and D are activated by the following line in log4ccp_solvem.conf :

```
priority hidden.features.ReducedOrderModeling DEBUG
```

The content of these matrices is described in D3.2. The information is essential to set up the coupling between the field solver and the circuit simulator.

6.4 priority hidden.features.MatrixTaylorExpansion DEBUG

With this priority, the result of the dump from 'hidden.features.dumpMatrixVFA' is decomposed in the various contributions originating from the time -independent term and the terms corresponding to the first-order and second-order time derivatives.

```
priority hidden.features.MatrixTaylorExpansion DEBUG
```

It should be noted that the matrices B, C and D are *a*lways generated as Taylor coefficients, since the sum $B_0 + B_1 d/dt$ has no use.

6.5 priority hidden.features.scaled_MTEXorROM_Dump DEBUG

With this flag we can control appearance of the desired output. If the priority is put on 'DEBUG', the matrices will be dumped as they are used inside the solver : the priority *e*xplicitly asks for the *s*caled values of the variables. One gets either all variables in scaled or all variables in non-scaled representation. If the priority is put different from 'DEBUG' (e.g. 'INFO') , then the matrices are expressed in SI units.

7 Transient module in MAGWEL

The python modules as described above can be launched with the MAGWEL solver. For that purpose it was desired to have an interface in place that in fully integrated in the MAGWEL environment. In this section, we describe the graphical user interface that is based on QT and Xerces (Xerces is a collection of software libraries for parsing, validating, serializing and manipulating XML. The library implements a number of standard APIs for XML parsing, including DOM. The MAGWEL software uses DOM (The Document Object Model is a cross-platform and language-independent convention for representing and interacting with objects in (X)HTML, and XML documents. Aspects of the DOM (such as its "Elements") may be addressed and manipulated within the syntax of the programming language in use.
The XML format that is applied to launch the transient simulation is given below. In particular, the 'options' part maps one-to-one on the options list as described by the DAEn solver usage.

```
<analysis>
  <static/>
  <time>
    <min>0</min>
    <max>1e-06</max>
```

```
      <steps>1</steps>
      <initialTime>0</initialTime>
      <mode>LIN</mode>
      <method>BDF1</method>
      <options>
        <abstol>1e-08</abstol>
        <reltol>0.0001</reltol>
        <initstep>0.001</initstep>
        <stepmax>0.01</stepmax>
        <ordmax>2</ordmax>
        <errorEstimationStrategy>1</errorEstimationStrategy>
      </options>
      <numberOfStepsBeweenOutputDumps>1
      </numberOfStepsBeweenOutputDumps>
    </time>
  </analysis>
```

The XML description is accessible through the GUI, as is illustrated below. DOM elements are entities that open and close with a key word such ¡option¿..¡/option¿. The content can be both elementary like a number or complex, like a subset of other DOM elements.
The documentation of the C++ file that facilitates the interfacing (API) is given below.

```
/***************************************************************************
                          cdstrtransientanalysis.cpp
                             -------------------
    begin                : May 25, 2008
    copyright            : MAGWEL
    author               : (C) 2008 by Gabor Bella
    email                : gabor@magwel.com
 ***************************************************************************
 *                                                                         *
 *   This is licensed software.                                            *
 *   This subroutine, function or program may not be sold or given away    *
 *   without the prior written consent of MAGWEL NV Martelarenplein 13     *
 *   B-3000 Leuven, Belgium                                                *
 *   @2008                                                                 *
 ***************************************************************************/

#include "cdstrtransientanalysis.h"

/**
  * Constructor. Takes the mode & method as integers
  * @param minimum  The minimum parameter.
  * @param maximum  The maximum parameter.
  * @param steps    The nr of steps.
  * @param outstep  The nr of steps to output.
  * @param mode   The step mode. This should be LINEAR, LOGARITHMIC,
  * for ADAPTIVE for respectively
  * linear, logarithmic, exponential or adaptive stepping.
  * @param method : solution method for time discretization
```

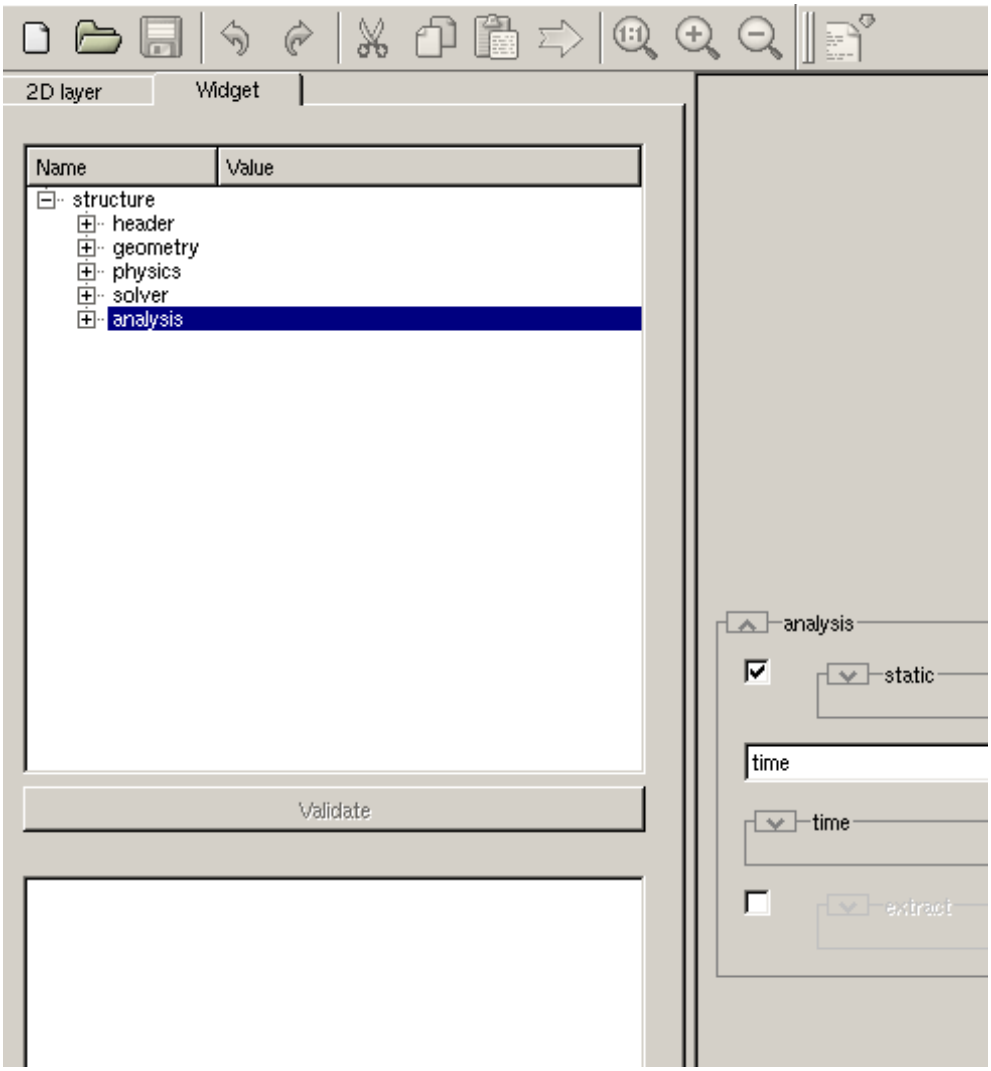

Figure 3: Main entrance to the transient GUI

```
* "BDF1" First order Backward-Differentation Formule and Implicit Euler
* "TRAP" Trapezium rule"
* "BDF2" Second order BFD with Implicit Euler
* "DAEn" Combined solver of BDF order and time step adaptation (Koln University)
* @param abstol = absolute tolerance (type real : default= 1.0e-3 )
* @param reltol = relative tolerance (type real : default= 1.0e-3 )
* @param stepmax = upper bound on the stepsize (type real : default= 1.0e-2 * t_len )
* @param initstep = initial stepsize (type real : default = 1.0e-3 * t_len )
* @param ordmax = used order of BDF  (type integer : 1-5 default 2 )
* @param errorEstimationStrategy (type integer : 0,1,2 )
* are all control parameters for the DAEn Koln solver
* @exception CdstrException  Gets thrown if the method is not correct.
*/
```

With these GUI we maintain a large flexibility to drive alternative implementations. In particular, the option menu allows for specific settings that are needed to deal with non-linear materials.

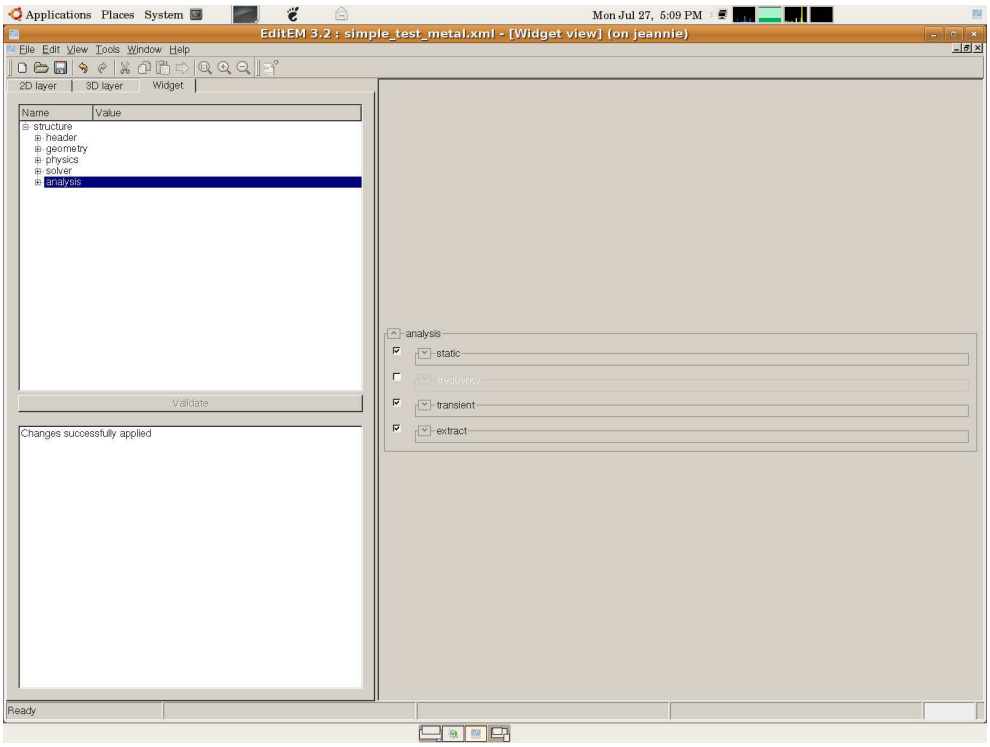

Figure 4: Main entrance to the transient GUI

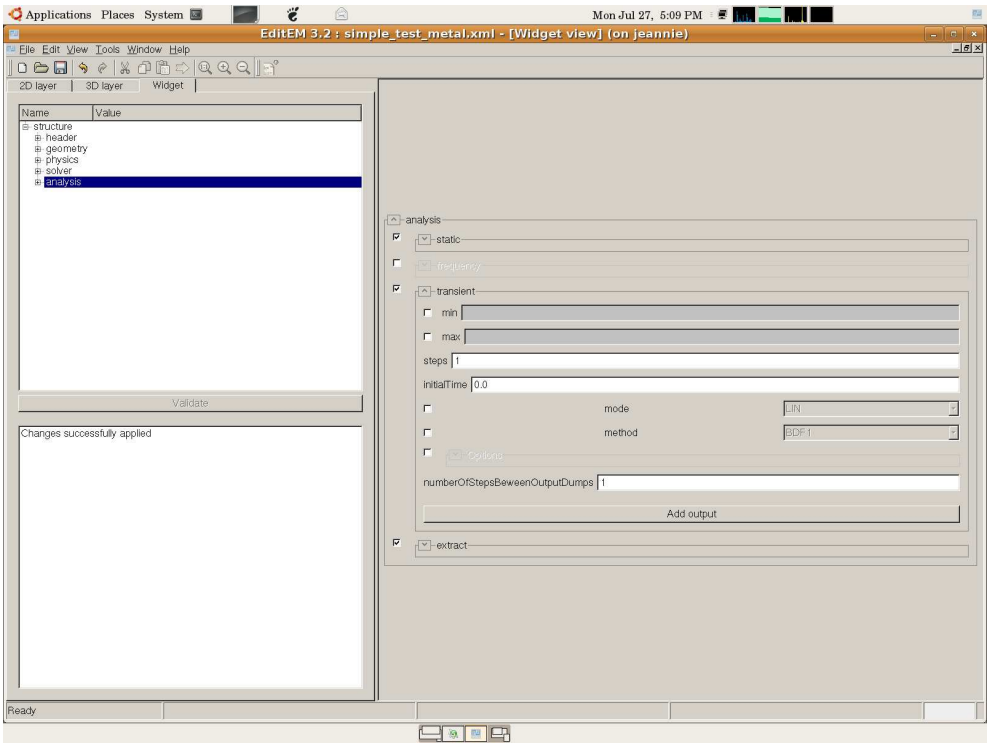

Figure 5: Transient menu details

Figure 6: Transient menu details

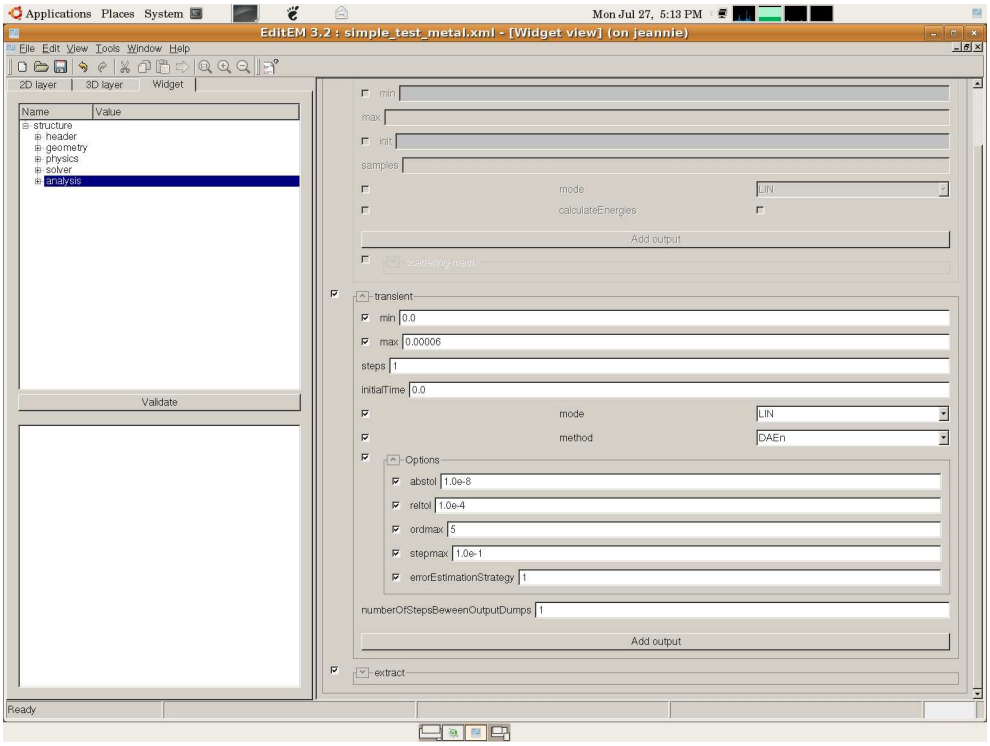

Figure 7: The menu for the DAEn options settings

8 Conclusions

In this deliverable we reported on the construction of the software modules that were foreseen in the WP3 activities. The goal of WP3 is to provide simulation support for exploring ranges of validity of the assumptions that underlie the circuit equations. Applications of the software modules will be presented in D3.4. Here, the primary goal is the report about the software developments of which the theoretical foundations are reported in D3.2. It should also be emphasized that D3.3 is primarily a *software* deliverable (not a report). As a consequence, we consider this report as "evidence" that the realization is software pursued.

The coupled EM circuit simulation is realized by a modular approach. It allows a transient simulation with different time and space discretization schemes. The coupling has been realized by a linear interface for linear problems as well as by a nonlinear interface for the general case. Both interfaces are running from the programming point of view. A number of technical problems had to be solved for this, for example duplicate memory allocations from the MECS solver and the MAGWEL solver or licence server issues.

The testing of the coupled simulation package is not yet finished. In D3.4 some test structures are described. Several problems have been shown up. Some of them could be solved, e.g. dimension and order questions for certain test elements or correct scaling factors. Others are still under investigation, e.g. correct function evaluations and Jacobian calculations in case of the nonlinear interface. One particular question to investigate is the consistency of the contact descriptions with the discretizations used for the transient simulation. It might be necessary to redesign the coupling realization for a successful treatment.

Finally, we note that a constructive approach to develop to theoretical ideas requires software development during their development. Contrary what is often advocated, new ideas and their implementation are done in a simultaneous fashion. Fragments of the software needs to be re-worked while the evolution of ideas progresses and the progression is driven by outcomes of running the software tools. The software, as reported here, differs from more conventional software projects in which one can clearly identify an analysis, a design, an implementation and a test phase and each phase is frozen before moving to the next stage. Nevertheless, we conclude that our efforts have led to operational tools as will be demonstrated at the final review.

References

[1] M. Hanke: *A New Implementation of a BDF method Within the Method of Lines*, Report No. 2001:01, Royal Institute of Technology, Stockholm, 2001.

[2] D. E. Schwarz, C. Tischendorf: *Mathematical Problems in Circuit Simulation*, Mathematical Modelling of Systems, Vol. 1, 1996.