ARGO

WCET-Aware Parallelization of Model-Based Applications
for Heterogeneous Parallel Systems

H2020-ICT-2015

Project Number: 688131

Deliverable D5.3

D5.3 End-user guidelines for model-based design

Editors:	Clément David (Scilab)
Authors:	Clément David (Scilab) Timo Stripf (Emmtrix) David Müller (DLR) Umut Durak (DLR) Marcus Bednara (IIS) Koray Kasnakli (IIS)
Version:	v1.00
Status:	FINAL
Dissemination level:	Public (PU)
Filename:	D5.3 End-user guidelines for model-based design.docx

ARGO Consortium

Karlsruhe Institute of Technology	DE
Scilab Enterprises	FR
Recore Systems B.V.	NL
Université de Rennes I	FR
Technological Educational Institute of Western Greece	GR
AbsInt Angewandte Informatik GmbH	DE
Deutsches Zentrum für Luft- und Raumfahrt	DE
Fraunhofer IIS	DE
emmtrix Technologies GmbH	DE

Document revision history

Version	Based on	Date	Author	Comments / Changes
V1.00		2017-07-17	Clément David	Final Version

About this document

This document provides end-user guidelines for application development in the model-based design environment. It accompanies the delivered implementations made by the software providers and also provides an integrated overview of the toolset. This deliverable is the first revision of the guidelines.

Table of Contents

List of Figures

List of Tables

No table of figures entries found.

1. Introduction

Models or applied maths on controls need arise at the end of the 20th century due to some challenges emergence:

1. How to control in safer ways critical infrastructures like nuclear plants?
2. How to ensure by design that suppliers will comply with specifications?
3. How to split a system and distribute the realisation of parts.

All these questions can be answered by providing a mathematical model backed by dedicated software called a “Model-based environment”. There are multiple mathematical representations as well as multiple environments available depending on users’ concerns. In our use-cases, the selected model-based environment is called Scilab/Xcos[1] and implements both a scripting language and a graphical dynamic systems modeler.

The two use-cases represents the industry usage on implementing advanced controls software in two different ways. The Fraunhofer IIS POLKA use-case focus on the easy to use, matrix-based Scilab scripting language to implement image transformation algorithms, whereas the DLR GPWS use-case rely on sub-systems decomposition and control system semantics of Xcos to implement an aeronautic altitude alert system.

2. Overview of ARGO model-based design toolset

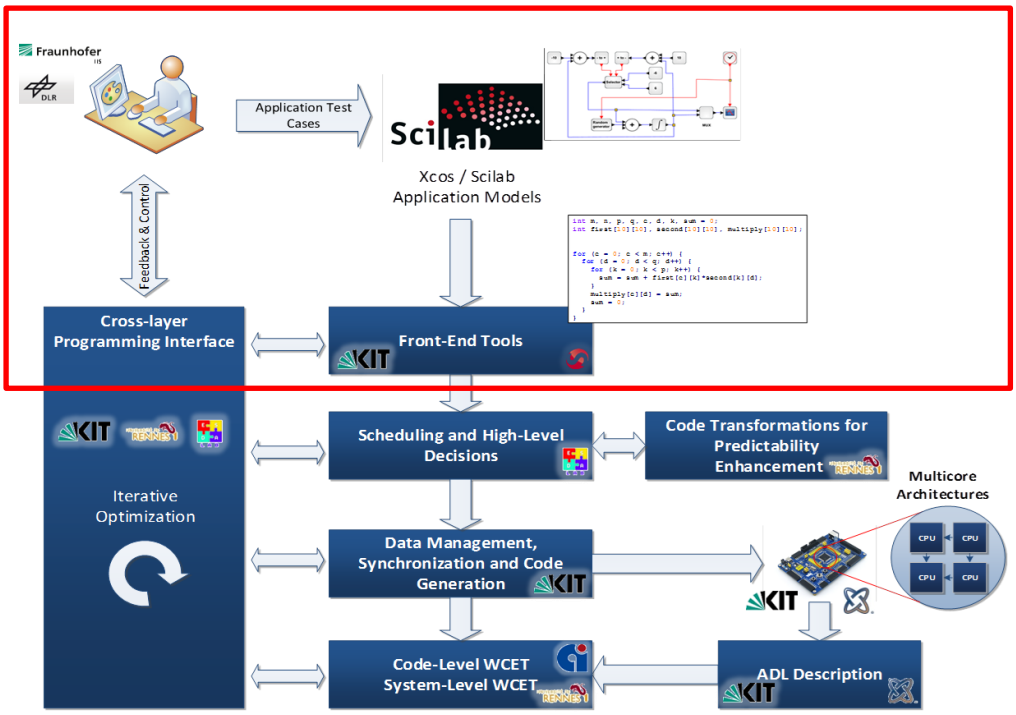


Figure 1: ARGO tool flow

In this document, the upper layer of ARGO tool flow will be detailed. The focus will be on user-visible toolset, how to edit the input model to feed the entire ARGO flow, get information back after a first iteration and specialize the input model to better fit the selected multi-core architecture.

3. Scilab/Xcos: a model based design environment

Scilab/Xcos is a generic environment that includes hundreds of mathematical functions. It has a high-level programming language allowing access to advanced data structures, 2-D and 3-D graphical functions. In the context of the ARGO project, the Scilab/Xcos define on high-level implementation, a model, validated on the design phase thanks to the dynamic evaluation of the Scilab language and the Xcos system modelling.

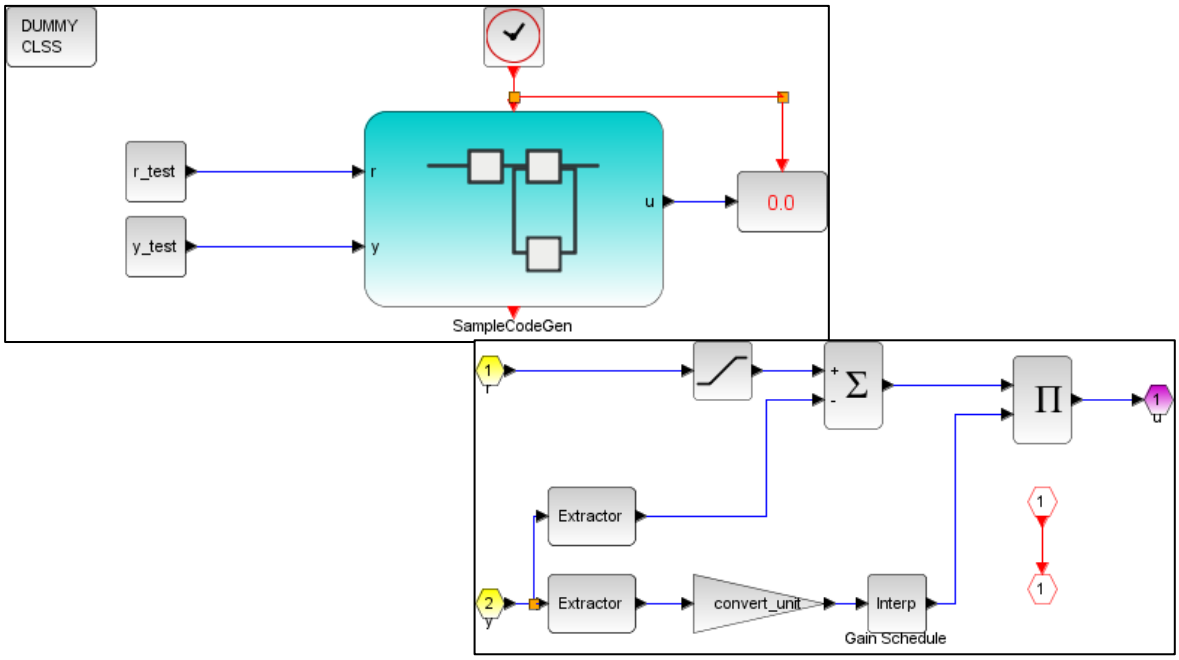


Figure 2: Xcos model

```
steptime = 0.01  // 1.0E-02

//constants
convert_unit = 3.28084          //m to ft conversion
r_satmax=[10]
r_satmin=[-10];

scheduling_state=1
controlled_state=2

//Inputs
r_test=10
y_test=[10,9,10,9,9,9]

//gain schedule
x1=[-50, 0, 50]       //state
y1=[1.12,0.40,2.45]   //gain
```

Figure 3: Scilab code as configuration

The high-level model could be defined using Scilab language with limited function-set, Xcos modelling with limited block-set or a mix of both as in Figure 2: Xcos model and Figure 3: Scilab code as configuration. The combination of having a scripting language to implement blocks as well as to parametrize a system definition is a key-requirement for the end-users. This will allow a unique way to match specific multicore architectures constraints by tuning and validating the model in a high-level toolset.

The software tools currently available are using Scilab 5.5.2 feature sets and require this exact version in the development machine. To be able to run the generated C code, a C compiler is also required.

3.1 Model design

Scilab is a matrix-based language offering native datatypes to solve mathematical, scientific, engineering problems. Xcos is graphical editor and simulator with precise semantics for computation-oriented design. Quick start guides[2] are available and provide a fast way to understand the language and its features.

Using Scilab, you can design a model by using the native datatypes provided and construct your own data models based on these native ones. The toolset has a support for most of the Scilab datatypes for building algorithms especially:

- Extended matrix support (scalar, vector, matrices and hypermatrix)
- `struct()` with named indexing
- Integers, float, double, string values

A detailed list of the supported functions with supported features and limitations is also available within the toolset[3]. Users should refer to this documentation to analyse if the subset used in its model is covered by the flow. The Scilab frontend has however been designed to be easily extendable by user-defined functions to cover the missing parts or implement stub used on validation.

Using Xcos, you can design a model by using the native functional blocks and define sub-systems by composition. The visual aspect can also be setup to add non-functional documentation and requirements. The toolset supports a subset of blocks as well as a subset of the modelling semantics; the available subset covers:

- Constant, mathematical and discrete blocks
- Scilab user-defined blocks
- Single or multi-synchronous event triggered subsystem

An initial list of supported blocks is shipped within the toolset and, as for the Scilab frontend, user-defined blocks can be implemented to add more features. Due to the WCET computation constraints and model real-time execution, the Xcos frontend only covers mono-clocked (or synchronized multi-clocked) blocks.

To facilitate the initial design of the model, a lot of books describing Scilab usage per scientific or engineering domains are already available. For example, the user can read books covering the whole Scilab software[4], practical problem solving[5] or how to port code from Matlab©[6]

During the development, the end users are also advised to follow the toolbox skeleton[7] to easily test their developments and share the model as a toolbox.

3.2 Model verification

After the initial design phase and during iterative development, the toolset can be used to check model correctness and compliance for code generation. The tool flow allows multiple verification at various transformation stages; each stage have a well-defined inputs and outputs feature sets checked during code generation. Issue found by a specific transformation stage could be linked back to the initial model implementation thanks to traceability links.

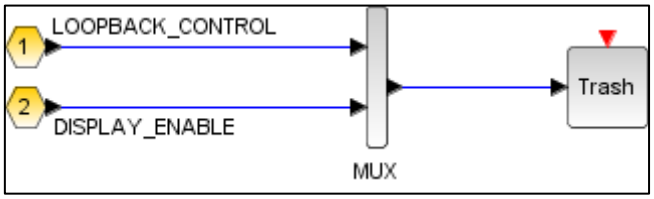


Figure 4: subsystem with a trash block

```
// 2aa0f3e1:15755feef64:-50b4
// scs_m(list("objs",1,"model","rpar","objs",33))
// PhysicalModel/ControlSubsystem
// multiplex
tmp155 = [ LOOPBACK_CONTROL ; DISPLAY_ENABLE ];
```

Figure 5: Scilab translated code

```
ControlSubsystem.sci(241:5-11): W00053 Warning: Variable tmp155 not used.
  tmp155 = [ LOOPBACK_CONTROL ; DISPLAY_ENABLE ];
  ~~~~~~
```

Figure 6: Scilab to C frontend warning for dummy assignment

Usual software verification such as unused variables are performed during the Scilab to C translation and can be considered as design flaws. For example, Figure 4, Figure 5, Figure 6 demonstrate the top-down toolset execution from an Xcos model to generated Scilab code and then to an “unused variable” warning emitted by the Scilab to C frontend.

The conversion from Scilab to C also performs an aggressive range analysis that can trigger warnings or partial code generation when the input model (either designed by hand or produced from Xcos) contains invalid parameters or constructs. Another example is to link together interpolation blocks with incompatible value ranges. This construct might not be detected on model design as Xcos and Scilab generated code will still simulate correctly. However, the Scilab to C frontend will display a warning as it is considered correctly as dead code.

Furthermore, models unit-testing facilities are also distributed with the toolset. Within Scilab or the Emmtrix software package the end-user can implement unit tests to check numerical correctness of the high-level model. During model design, independent subsystems or functions can then be verified.

Model validation, e.g. model accuracy evaluation, can also be performed using the toolset, however it is left to the user to develop validation methods corresponding to its design and practices. From our experience, the ARGO toolset is flexible enough to match either Xcos

subsystem validation as in the DLR EGPWS use-case, either Scilab gateway stub validation as used in the IIS POLKA use-case.

3.3 Software in the loop (SIL)

Quoting ACM SIG on SImulation and Modeling[8] :

> *Software-in-the-loop can be viewed as Simulation-based Software Evaluation. A software system can be executed under simulated input conditions for the purpose of evaluating how well the software system functions under such input conditions.*

In the context of the ARGO project, Software in the loop could be performed to ensure model consistency across the different transformation performed by the tools. Each tool, composing the toolset, has well-defined inputs, outputs and computation function that can be used to re-inject the tool output back to the user-defined main model.

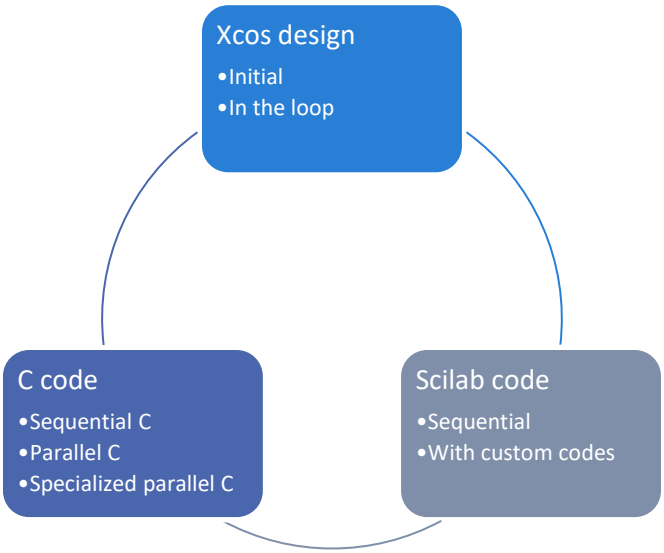


Figure 7: ARGO tools in the loop

Depending on the user needs, the toolset offer a wide range of capabilities to reinject generated computation functions back to the original model as shown in Figure 7. For example:

1. Starting from an Xcos model, the generated Scilab code can be re-injected back as a Scilab function block.
2. From a Scilab model, the generated sequential C code can be compiled using Scilab facilities and called as a host native implementation.
3. Using the same Xcos model as in 1., the generated (sequential or parallel) C code can be introduced back to the model as a C native function block.

In a more advanced topic, platform specific parallelization support might also be checked by reinjecting back not only the generated sequential code but also multiple parallelized versions or, even better a hardware simulation to ensure correctness of the final behaviour.

Starting from Software in the loop, the end-user could also want to go deeper using in-the-loop validation (with Processor in the loop, Hardware in the loop, complex physical models in

the loop). Thanks to Scilab features and to the large user-contributed toolboxes, this advanced model checking can be performed with ease and have already been demonstrated by IIS by accessing a remote hardware.

4. Development iterations

During the first part of the ARGO project, end-users constructed models based on the existing documentation. As Scilab/Xcos is a previously existing software, the end-users started early and quickly by defining their models with support from software providers. A dedicated workshop also helped them to review their initial design.

To ease complete code generation, partial models should regularly be used as inputs to the toolset to perform model verification. The toolset can then be used as a model-checking tool to ensure both toolset functional coverage and subsystems (or sub-functions) unit testing. Such an iterative approach successfully led to DLR GPWS and IIS POLKA C code generation.

As the toolset is currently top-down, the use cases design can be decomposed to a top-down transformation steps that should be drived by the end-user to correctly emit parallel C code for the chosen target architecture.

4.1 End-user application initial validation

Depending on the chosen modelling language used for the design, the initial validation differs:

- For a Scilab model, the validation should be performed through classical software testing.
- For an Xcos model, the validation should be performed by simulating the designed sub-system in a representative global system.

4.1.1 Scilab-based models validation

A Scilab test file is an executable script file with specific attributes as comments to setup the test environment correctly. Each test is executed inside its own test environment and dedicated Scilab runtime. To ease user checking, the test can be launched directly within Scilab using the `test_run()` utility function.

A notable point for validating Scilab based models is that the Scilab language is a dynamic-typed language which resolves functions at runtime and can locally re-define functions as showcased in Figure 8: Scilab function re-definition. This function redefinition allows unit-testing using stubs or more advanced mocking pattern.

```
function foo()
    bar(1, 2)
endfunction

bar = disp;
foo()

clear bar;
bar = sum;
foo()
```

Figure 8: Scilab function re-definition

The unit-testing of software components has already been vastly studied on the Computer Science literature[9] and is not covered in this document. Most of the described technics in the literature can also be applied to the Scilab language.

4.1.2 Xcos-based model validation

Xcos model validation is performed through the usual sub-system decomposition and the implementation of a representative external environment. The external environment is usually also defined using Xcos blocks but is not limited to the block subset supported on code generation.

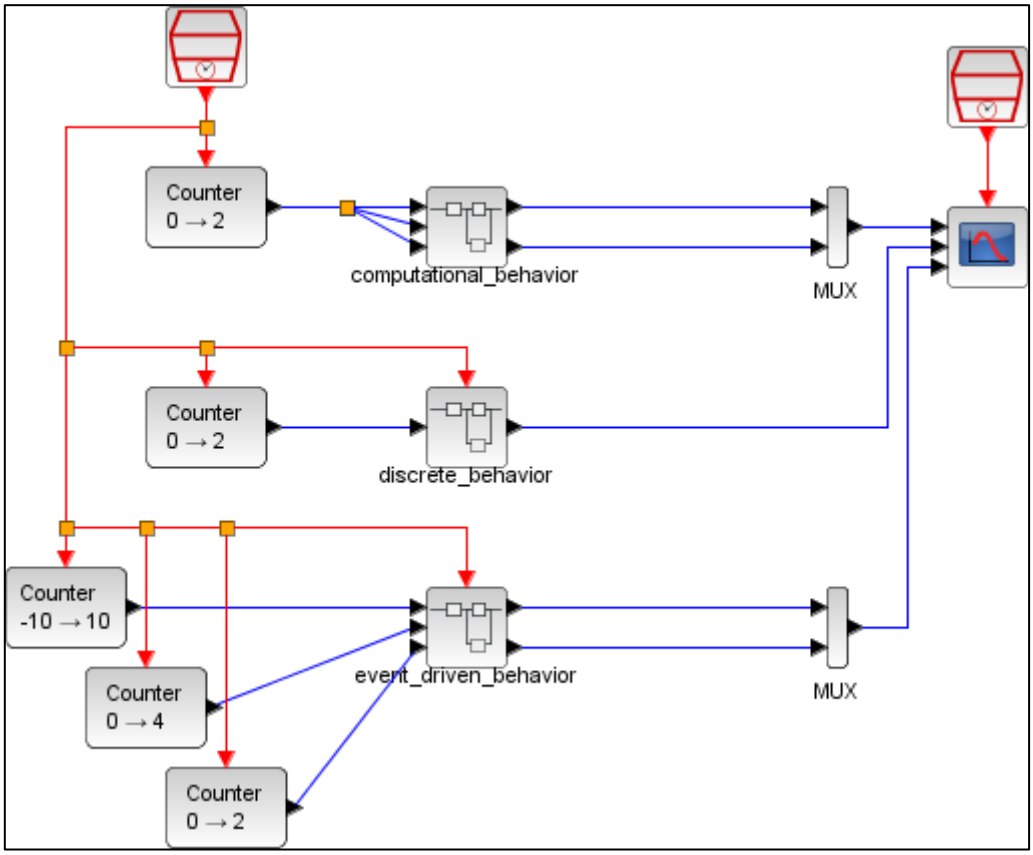


Figure 9: Xcos model for validation

```
// load xcos
loadXcosLibs();
// import the schema
importXcosDiagram(path + "/behaviors.zcos");

// generate some code for the superblock on TMPDIR
ok = xcg_codegenerator(scs_m, TMPDIR, "computational_behavior");

// execute and display the generated code
exec(TMPDIR + filesep() + name + ".sci", 1);

// convert the code to C using the 2nd frontend on TMPDIR
```

```
cd(TMPDIR)
emx_codegen(name + "_scenario.sce");
```

Figure 10: Xcos model test

The Figure 10: Xcos model test can also be extended to compile and link the generated code into Scilab and perform a Software in the loop (SIL) tests within the same graphical environment used during the design. Numerical correctness could then be ensured between the design model and generated code.

Note that in the first revision of the toolset, the capability to reinject the generated code back to the end-user model automatically as described in Software in the loop (SIL) is not available. Arguments passing and compilation should be done by hand for representative models. The arguments checking and compiler script generation is planned in future revision of the toolset within the integrated toolchain.

After this Xcos models can use Scilab variables to parametrize any block, the overall schema is usually setup to have one simulation but can be configured to simulate the same schema in a different context, with different input test vectors thus covering more executions in a single test.

4.2 Software in the Loop validation

Using the ARGO toolset, the end-user can validate the functional correctness of the generated code on different steps. Individual steps could be used to ensure generated versus model functional match and they could also be composed from model to targeted generated code from end-to-end functional validation.

The SIL validation of each transformation step should be performed inside the input model to avoid misevaluation due to executing the generated code inside a different execution environment (or software). For example, to validate:

1. generated Scilab code from an Xcos model, the Scilab code should be reinjected back into the Xcos model,
2. C code from Scilab, the C code should be compiled, linked and executed within Scilab,
3. Generated C code from (intermediate Scilab code from) Xcos model, the C code should be compiled, linked and executed as an Xcos block.

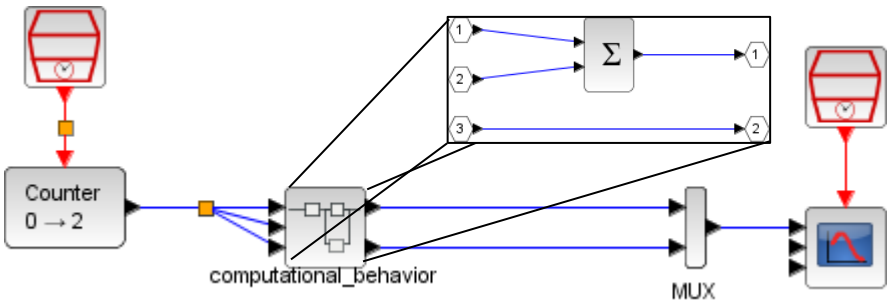
Currently the integrated toolchain runs top-down; e.g. from the model to the generated C code. The in-the-loop validation should be performed manually by the end-users. An automatic “stub” generation is planned to ease SIL implementation back to the input model (for each intermediate representation).

Each intermediate representation composing the toolset is described below as *INPUTS* and *OUTPUTS* of each tool composing the toolset. These intermediate representations can then be composed to perform one-step or multi-step model validation. Each “quoted” asset below is supposed to be opaque to the end-users and is only generated to ease tool flow iterations.

4.2.1 Xcos to Scilab

INPUTS: Xcos user-defined model, Scilab user-defined functions, Scilab user-defined scripts.

OUTPUTS: Scilab user-defined functions, Scilab generated functions, Scilab "scenario" script.




```
my_sum_parameters = [1 ; 1 ]
```

Figure 11: Xcos frontend input model for SIL

```
function [y1, y2]=computational_behavior(t, u1, u2, u3)
    //EMX?: emx_func_file('computational_behavior.c')


    // update outputs

    // 7b556cf2:1571d232759:-7da2
    // scs_m(list("objs",3,"model","rpar","objs",6))
    // behaviors/computational_behavior
    // sum
    tmp16 = u1 + u2;

    // 7b556cf2:1571d232759:-7da4
    // scs_m(list("objs",3,"model","rpar","objs",19))
    // behaviors/computational_behavior
    // xcg_output_sim
    y2 = u3;

    // 7b556cf2:1571d232759:-7da6
    // scs_m(list("objs",3,"model","rpar","objs",17))
    // behaviors/computational_behavior
    // xcg_output_sim
    y1 = tmp16;
endfunction
```

Figure 12: Xcos frontend output for SIL

Xcos have some blocks to use Scilab (using `scifunc_block_m`) or C code (using `c_block`) as a simulation function, re-injecting the generated Scilab code (or any generated C code) is simple and is planned to be done automatically in future revision of the toolset. More complex software in the loop implementations, such as remote execution through Ethernet / UDP network, have also been demonstrated successfully on the DLR use-case.

Using Xcos as an input environment also opens the door to more complex modelling of the outer environment. Representative plane dynamics or high-speed ground variation (multi-

clocked) can then be implemented using the full Xcos semantics on the model; still using the reduced semantic on the sub-system.

4.2.2 Scilab to sequential C

INPUTS: Scilab user-defined functions, Scilab generated functions, Scilab "scenario" script.

OUTPUTS: C generated functions, HTML report, C "main" code.

```
// initial output values
y1 = zeros(1, 1);
y2 = zeros(1, 1);

// initial input values
u1 = zeros(1, 1);
u2 = zeros(1, 1);
u3 = zeros(1, 1);

// initialize all the blocks

// simulation loop
for t=1:1:30
    // generating random inputs
    u1 = 1.797693134862315700000000000E+308 * rand(1, 1) -
8.988465674311578500000000000E+307;
    u2 = 1.797693134862315700000000000E+308 * rand(1, 1) -
8.988465674311578500000000000E+307;
    u3 = 1.797693134862315700000000000E+308 * rand(1, 1) -
8.988465674311578500000000000E+307;

    [y1, y2] = computational_behavior(t, u1, u2, u3);

    // print out values
    disp(y1);
    disp(y2);
end
```

```
function [y1, y2]=computational_behavior(t, u1, u2, u3)
    //EMX?: emx_func_file('computational_behavior.c')


    // update outputs

    // 7b556cf2:1571d232759:-7da2
    // scs_m(list("objs",3,"model","rpar","objs",6))
    // behaviors/computational_behavior
    // sum
    tmp16 = u1 + u2;

    // 7b556cf2:1571d232759:-7da4
    // scs_m(list("objs",3,"model","rpar","objs",19))
    // behaviors/computational_behavior
    // xcg_output_sim
    y2 = u3;
```

```
    // 7b556cf2:1571d232759:-7da6
    // scs_m(list("objs",3,"model","rpar","objs",17))
    // behaviors/computational_behavior
    // xcg_output_sim
    y1 = tmp16;
endfunction
```

Figure 13: Scilab frontend input for SIL

```
void computational_behavior(double * const y1_data, double * const y2_data,
double u1_data, double u2_data, double u3_data) {
  double tmp16_data;

  // computational_behavior.sci(2:13-56):
emx_func_file('computational_behavior.c')
  // computational_behavior.sci(11:5-21):  tmp16 = u1 + u2;

  tmp16_data = u1_data + u2_data;

  // computational_behavior.sci(17:5-13):  y2 = u3;

  *y2_data = u3_data;

  // computational_behavior.sci(23:5-16):  y1 = tmp16;

  *y1_data = tmp16_data;
}
```

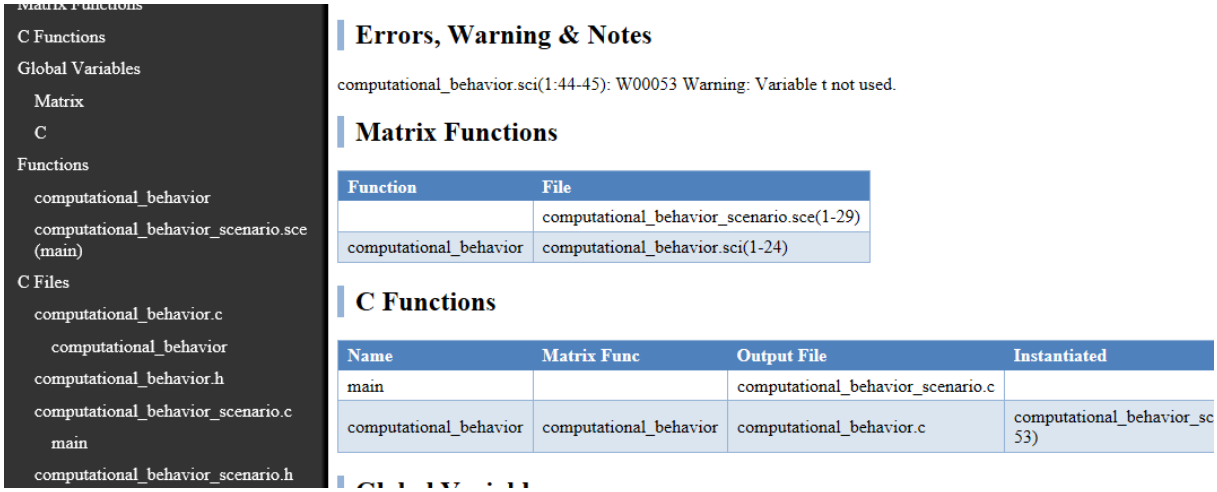


Figure 14: Scilab frontend output for SIL

Scilab can dynamically compile, link and execute C code using `link` and `call` functions to re-inject the generated C code back to Scilab. The generated code can then be stressed using all Scilab environment features; including graphical features, test vector generation or reporting.

As an added feature, the Scilab to C tool also comes with a comprehensive tests battery which can be used to implement unit-tests or non-regression tests for code snippets.

Numerical correctness of functions can then be tested out of the main Scilab environment in an automated way.

5. Conclusion

In this document, we described the ARGO toolset model-based design guidelines to create models and generate sequential C from it. The frontend part of the ARGO toolset provides facilities to validate a design and rely on code-generation to actually implement it. Two use-cases have been used in the context of the project to validate the approach and the usability of the tools.

Thanks to the holistic approach of the toolset, the use-cases will be used as input for automatic WCET-aware parallelisation in the next months and a second iteration of the guidelines will describe how to tune the input models to better fit the end-user requirements.

Glossary of Terms

ADL	Architecture Description Language
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric MultiProcessing
API	Application Programming Interface
ASG	Abstract Syntax Graph
AST	Abstract Syntax Tree
AVX	Advanced Vector Extensions
BUG	Bottom-Up-Greedy
CDFG	Control and Data Flow Graph
CFG	Control Flow Graph
CIL	Common Intermediate Language
CPU	Central Processing Unit
CRISP	Cutting edge Reconfigurable ICs for Stream Processing
DSL	Domain-Specific Language
DTSE	Data Transfer and Storage Methodology
GCC	GNU Compiler Collection
GPP	General Purpose Processor
GSP	General Streaming Processor
DSP	Digital Signal Processor
ELF	Executable and Linking Format
EULA	End User Licence Agreement
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
HDL	Hardware Description Language
HIR	High Level Intermediate Representation

HLS	High Level Synthesis
HPC	High Performance Computing
IMS	Integrated Modulo Scheduling
IR	Intermediate Representation
ISA	Instruction set architecture
JIT	Just In Time
LIR	Low Level Intermediate Representation
LISA	Language for Instruction Set Architecture
LISP	Is a family of computer programming languages
LLVM	Low-Level Virtual Machine
LTI	Linear Time-Invariant
MCA	Multicore Association
MMX	Multi Media Extension
MPSoC	Multiprocessor System on Chip
MPPB	Massively Parallel Processor Breadboarding
NLP	Nested Loop Programs
NP	Non-Polynomial
NoC	Network on Chip
NUMA	Non-Uniform Memory Access
OpenCL	Open Computing Language
PCCA	Partial Component Cluster Assignment
PIS	Pragmatic Integrated Scheduling
PIP	Parametric Integer Programming
PTX	Parallel Thread eXecution
RHOP	Region-based Hierarchical Operation Partitioning
RTL	Register Transfer Level
RFD	Reconfigurable Fabric Device

SCoP	Static Control Part
SIMD	Single Instruction Multiple Data
SMP	Symmetric MultiProcessing
SoC	System-on-Chip
SSA	Static Single Assignment
SWP	Sub-Word Parallelism
UAS	Unified Assign and Schedule
UMA	Uniform Memory Access
VHDL	Very high speed integrated circuits Hardware Description Language
VLIW	Very Long Instruction Word

References

[1] https://www.scilab.org a Free and Open-Source Software for numerical computation

[2] http://www.scilab.org/resources/documentation/tutorials

[3] Emmtrix ECG Reference Guide, ECGReferenceGuide.pdf

[4] Campbell, S. L., Chancelier, J. P., & Nikoukhah, R. (2010). Modeling and Simulation in SCILAB. *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*, 73-106.

[5] https://www.d-booker.fr/scilab-book1/325-scilab-fundamentals.html

[6] http://www.scilab.org/resources/documentation/community

[7] https://wiki.scilab.org/howto/Create%20a%20toolbox

[8] http://www.acm-sigsim-mskr.org/MSAreas/InTheLoop/softwareInTheLoop.htm

[9] https://en.wikipedia.org/wiki/Unit_testing#Notes