Integration of Tool Chain Extension Modules with the COE

Deliverable Number: D5.3a
Version: 1.0
Date: 2017
Public Document

http://into-cps.au.dk
Contributors:

Carl Gamble, UNEW
Victor Bandur, AU
Oliver Möller, VSI

Editors:

Carl Gamble, UNEW

Reviewers:

Julien Ouy, CLE
Etienne Brosse, ST
Frederik Földager, AI

Consortium:

<p>| Aarhus University | AU | Newcastle University | UNEW |
| University of York | UY | Linköping University | LIU |
| Verified Systems International GmbH | VSI | Controllab Products | CLP |
| ClearSy | CLE | TWT GmbH | TWT |
| Agro Intelligence | AI | United Technologies | UTRC |
| Softeam | ST | | |</p>
<table>
<thead>
<tr>
<th>Ver</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>2017-08-25</td>
<td>Carl Gamble</td>
<td>Skeleton Structure</td>
</tr>
<tr>
<td>0.1</td>
<td>2017-10-24</td>
<td>Carl Gamble</td>
<td>Contributors and Reviewers added</td>
</tr>
<tr>
<td>0.2</td>
<td>2017-10-30</td>
<td>Carl Gamble and Oliver Möller</td>
<td>DSE and MC/TA added</td>
</tr>
<tr>
<td>0.3</td>
<td>2017-10-31</td>
<td>Victor Bandur</td>
<td>Code Generation added</td>
</tr>
<tr>
<td>0.4</td>
<td>2017-10-31</td>
<td>Carl Gamble</td>
<td>Title corrected</td>
</tr>
<tr>
<td>0.5</td>
<td>2017-11-02</td>
<td>Carl Gamble</td>
<td>Introduction added, ready for review</td>
</tr>
<tr>
<td>1.0</td>
<td>2017-12-12</td>
<td>Carl Gamble</td>
<td>Internal review comments addressed</td>
</tr>
</tbody>
</table>
Abstract

The deliverable is limited to highlighting the interactions between the tool chain extension modules and the COE and INTO-CPS Application. These modules include design space exploration, model checking and test automation, and code generation. Details of the functionality of these modules may be found in deliverables D5.3e DSE [Gam17], D.3c ‘Implementation of a Model Checking Component for Global Model Checking’ citeINTOCPSD5.3c, and D5.3d ‘FMI-Compliant Code Generation in the INTO-CPS Tool Chain’ [BHPG17].
Contents

1 Introduction 6

2 DSE 6
   2.1 Configuration and Launch of DSE 7
   2.2 Simulation on local machine 7
   2.3 Simulation on Cloud services 8

3 Model Checking and Test Automation 11

4 Code Generation 13

5 Conclusions 13

6 List of Acronyms 14
1 Introduction

When the project description of action (DOA) was penned, this deliverable was added to describe how the modules developed in work package 5 had been integrated ‘into’ the COE. At the time the DOA was written it was not clear exactly what form the final INTO-CPS tool chain would take and so it was considered that the work package 5 modules could actually be packaged in or compiled into the COE, however, in the end the tools maintained their own identity in their own packages and so were not ‘integrated’ into one package as such. Instead, this deliverable describes the interactions between the Work Package 5 modules and both the COE and the INTO-CPS Application. The structure of the deliverable starts with the design space exploration (DSE) scripts from task T5.1, then presents the model checking and test automation support from tasks T5.2 and T5.3 before concluding with a brief statement about code generation from T5.4.

2 DSE

The DSE functionality exists as a collection of python scripts and so is not compiled into INTO-CPS application or the COE, there are, however, links between these three. Essentially, the INTO-CPS application is able to create DSE configuration files and then make use of the DSE scripts to perform DSE, then the DSE scripts in turn make use of the COE to actually run the simulations, Figure 1.

The following subsections outline the interactions with the DSE scripts. It starts with the interaction with the INTO-CPS application and then describes the two different interactions that take place with the COE depending on whether DSE will make use of the user’s local machine to run simulations or if they are making use of the cloud. The latter two sections only look at how the COE is used, for full details on the different operating modes for
2.1 Configuration and Launch of DSE

The interactions between the INTO-CPS application and the DSE scripts have two distinct elements, Figure 2. The first element is the editing of DSE configuration files, these configuration govern all aspects of the DSE process including the design parameters to sweep over, the choice of search algorithm, the means by which designs are evaluated and ranked, and several others. Details of how DSE configurations are created and edited may be found in the user manual, D4.3a [BLL+17].

The second interaction between the application and the DSE scripts is the launching of the DSE process. Since the DSE scripts are not compiled into the application and are standalone Python scripts, once a user is happy with a configuration and they click the launch button, the application then launches the DSE scripts, passing them the required arguments so they may find the INTO-CPS project workspace and configuration files. Details of how DSE is launched from the application may be found in, D4.3a [BLL+17], while details of the command line arguments passed to the scripts may be found in D5.3e [Gam17].

2.2 Simulation on local machine

When running DSE on the user’s machine, the DSE scripts assume that there is an instance of the COE running and that it has the permissions needed.
to expand FMUs as needed. As such, the scripts do not need to launch the COE or interact directly with the FMUs.

The interaction between the DSE scripts and the COE is shown in Figure 3. The outline sequence is that the DSE scripts request a session number from the COE, it then uses the sessions key when transmitting the simulation configuration to the COE, launching the simulation and finally retrieving the raw simulation results.

The interaction sequence is implemented in a single Python script, the `coehandler.py`, making use of the `curl` application to interact with the COE’s http interface.

### 2.3 Simulation on Cloud services

As described in D5.3e on DSE [Gam17], scripts have been developed that permit the running of multiple parallel simulations runs by making use of the HTCondor software system. While the governance of the DSE process takes place on a local machine, the HTCondor computation nodes may be deployed on cloud services with as many replications as are needed.

When running a DSE in the cloud we face a different environment and this affects how the simulation process is controlled. When the search algorithm has determined the simulations that should be run, these are distributed among the available compute nodes and the execution on each node is controlled by a Windows batch file. An example of such a batch file is shown in Figure 4 and the process is outlined graphically in Figure 7. The use of a...
Figure 4: An example of the batch file controlling interaction with the COE on the cloud

```bash
C:\COE\java64\1.8.0_102\bin\jar.exe sf config_package.zip
C:\COE\java64\1.8.0_102\bin\jar.exe sf simulation_package.zip
C:\COE\java64\1.8.0_102\bin\jar.exe sf analysis_package.zip

copy %CD%\config_package\config21.mm.json %CD%\simulation_package\config.mm.json

od simulation_package
6
copy %CD%\analysis_package\analysis_package L2 %CD%\analysis_package\studentMap time (bodyFMU).body.robot_x (bodyFMU).body.robot_y

Figure 5: An example of the batch file controlling interaction with the COE on the cloud

Windows batch file in dictated in the case of the example by the deployment of the HTCondor compute nodes onto Windows based hosts. The compute nodes could be deployed onto Linux or OS X, in which case the batch file would need to be altered accordingly.

The first task of the batch file is to expand an archive file containing extracted FMUs and the ‘objective programs’ that compute the objective values (Figure 4 lines 1 – 3). The extracted FMUs are necessary since the permissions given to condorHT on the compute nodes do not allow any process to write to the normal ‘temp’ folders. To work around this we make use of a COE feature that means if, in the multi-model configuration file, it is passed a folder name (Figure 5) rather than an FMU file name (Figure 6) it will assume that folder contains an already extracted FMU. The HTCondor compute nodes used in the example did not have Python installed and so the objective programs used here take the place of the normal Python scripts that compute simulation objective results when running simulations locally.

The second step is to copy the required multi-model configuration file so it is the correct location for the COE to read (Figure 4 line 5) (Details on why
this is required may be found in D5.3e).

With the FMUs and objective programs extracted, the batch file may then launch the COE in ‘one shot’ mode (Figure 4, line 9). This is a special mode where the COE is passed the path to a file containing the simulation configuration and also the path the file it should create to store the simulation results.

Finally, the batch file is responsible for executing the programs that process the raw simulation result to compute the objective values (Figure 4, lines 13 – 15). Once the objective values are stored in a JSON formatted file, the batch file completes, this triggers the exit behaviour of the condor scripts that transfers the simulation results back to the host controlling the search. The user has the option of returning either just the objective results or objective results and the raw simulation results by adjusting a script governing submission of ‘jobs’ to the HTCondor system.
3 Model Checking and Test Automation

For test automation (TA), the COE is the central engine that executes a test run (i.e., a COE experiment).

By convention, the first FMU that contributes to this run is the test driver, which defines the stimuli and performs the check operation that determine the result of the test execution.

A test execution requires the following ingredients.

1. a test FMU
2. one or more FMU constituting the system under test
   (some may be simulations or generated simulations here)
3. a step size
4. the test duration (timeout).

As for the test duration, the test FMU also should determine this, since it needs to finish the sequence of prepared stimuli. Therefore the value for
this should usually be “auto”, i.e., determined by the “default experiment duration” declared by the first FMU.

Figure 8 shows how to invoke a test execution in the example project “three water tank”. The screen-shot is taken from the RT-Tester graphical user interface; alternatively, this operation can also be started from the INTO-CPS Application.

The important utility that is installed together with the examples is the python script “run-COE.py”. The command above is expanded to a script invocation as shown in Figure 9.

```bash
../utils/run-COE.py RTT_TestProcedures/TP-BCS
   RTT_TestProcedures/SUT1
   RTT_TestProcedures/SUT2
   RTT_TestProcedures/SUT3
   RTT_TestProcedures/SUT_controller
   --timeout auto
   --stepsize 0.01
```

Figure 9: Translation of the user dialogue to Script invocation.

For typical test projects, all the FMUs are created within the RT-Tester tool (either as SUT wrapper, as Simulation, or as Mock-up). In this case, the name and type of the interface variables is defined by the test model and therefore unique for all involved FMUs. Then the connection diagram can be automatically derived by the utility script “run-COE.py”.

In situations where one or more FMUs originate from another context, this mapping can be defined explicitly via a JSON formatted file. “run-COE.py” then requires the additional command line option

```
--connections=JSON_FILE_WITH_DEFINITIONS
```

The full list of command line options is listed in Figure 10.

The COE is not directly involved with evaluation of the test results - this is performed by RT-Tester mechanisms that inspect the output of the test FMU. For example, the observed behavior with respect to some model elements is compared to the expected behaviour. For matches, a PASS is generated and for mismatches a FAIL. Unreached situations remain INCONCLUSIVE. This evaluation can be mapped to the connected (SUT-)requirements. This is shown in Figure 11.

For Model-Checking operations, the COE is not involved.
4 Code Generation

The COE makes use of code generation [BHPG16] only indirectly. The simulation tools Overture, 20sim and OpenModelica can all export their models to C. This code is wrapped with an FMI-compliant layer, producing standalone FMUs. The code is compiled for the platform on which the COE is executing, and the resulting FMU can be used as a drop-in replacement for the corresponding tool-wraper FMU. During co-simulation, the COE interacts with standalone FMUs in the same way as with tool-wraper FMUs, but because they are compiled and not interpreted, standalone FMUs usually execute much faster.

5 Conclusions

This deliverable was originally intended to show how the tool-chain modules were integrated with the COE, however, as described the integration of the modules did not take place in the way envisaged when the project description of action was constructed. Instead the deliverable has described how those modules make use of the COE and are made use of by the INTO-CPS Application.
References


6 List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AU</td>
<td>Aarhus University</td>
</tr>
<tr>
<td>CLE</td>
<td>ClearSy</td>
</tr>
<tr>
<td>CLP</td>
<td>Controllab Products B.V.</td>
</tr>
<tr>
<td>COE</td>
<td>Co-simulation Orchestration Engine</td>
</tr>
<tr>
<td>DSE</td>
<td>Design Space Exploration</td>
</tr>
<tr>
<td>ENUM</td>
<td>Enumeration and Scoring</td>
</tr>
<tr>
<td>FMU</td>
<td>Functional Mockup Unit</td>
</tr>
<tr>
<td>PROV-N</td>
<td>The Provenance Notation</td>
</tr>
<tr>
<td>ST</td>
<td>Softeam</td>
</tr>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>TA</td>
<td>Test Automation</td>
</tr>
<tr>
<td>TWT</td>
<td>TWT GmbH Science &amp; Innovation</td>
</tr>
<tr>
<td>UNEW</td>
<td>University of Newcastle upon Tyne</td>
</tr>
<tr>
<td>UTRC</td>
<td>United Technology Research Center</td>
</tr>
<tr>
<td>UY</td>
<td>University of York</td>
</tr>
<tr>
<td>VSI</td>
<td>Verified Systems International</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>WAM</td>
<td>Weighted Additive Method</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
D5.3a - Integration of Modules with the COE (Public)

Usage: run-COE.py TestProc1 TestProc2 [TestProc3, ... ]

Starts an test execution with two FMUs that are (from RT-Tester perspective) RTT6 test procedures wrapped in FMUs.

TestProc1 : The test driver (directing stimulation and checking)
TestProc2,3,... : The system under test (SUT) which is compared to the expected behaviour

The COE will check whether all inputs/outputs fit together; it is a user obligation to construct the TestProcs such that the corresponding FMU interfaces constitute a closed system.

Options:
- --version show program’s version number and exit
- -h, --help show this help message and exit
- -t DURATION, --timeout=DURATION
  define the duration of the run (in seconds). If this value is set to 'auto', then the duration is taken from the DefaultExperiment of the first FMU (+ 1.0 second slack added).
- -s DURATION, --stepsize=DURATION
  define the step size the COE shall use (in seconds), default: 0.1. If this value is set to 'auto', then the step size is taken from the DefaultExperiment of the first FMU.
- -p PORT, --port=PORT define the port to connect with the COE
- -c, --query-coe-version
  Query the version of the COE, display it and exit
- -i FILE, --io-config=FILE
  Point to an override *.json file that defines the COE configuration; needs to map "connections",
- -C FILE, --connections=FILE
  Point to an override *.json file that defines connections as connections["<input>"] = [ <output>* ] The <input>/<output> is structured as fmuName.instanceName.varName The data from this file will be used in the COE run *instead* of the derived connections[]. The place holders @GUID_TP1@, @GUID_TP2@, ... can be used to reference the respective GUID of the respective TestProc.
- --verbose Print all debugging output

Figure 10: Command line options to modify the COE invocation.
Figure 11: RT-Tester Test-Automation result, requirements may be PASS, FAIL, or INCONCLUSIVE (i.e., untried).