Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs

INTO-CPS Traceability Implementation

Deliverable Number: D4.3d

Version: 1.0

Date: December 2017

Public Document

http://into-cps.au.dk

Contributors:

Kenneth Lausdahl (AU)
Jos Höll (TWT)
Christian König (TWT)
Carl Gamble (UNEW)
Oliver Möller (VSI)
Etienne Brosse (ST)
Tom Bokhove (CLP)
Luis Diogo Couto (UTRC)
Adrian Pop (LIU)
Alachew Mengist (LIU)

Editors:

Christian König (TWT)

Reviewers:

Ken Pierce (UNEW)
Kangfeng Ye (UY)
Stylianos Basagiannis (UTRC)

Consortium:

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		

Document History

Ver	Date	Author	Description
0.1	19-07-2017	C. König (TWT)	Initial document version
0.2	19-10-2017	C. König (TWT)	Updated structure and content
0.3	09-11-2017	C. König (TWT)	Incorporated comments from internal review
1.0	14-12-2017	C. König (TWT)	Updated queries with new features, ready for submission

Abstract

This deliverable covers the implementation of the traceability and model management features in INTO-CPS. The implementation builds on top of the design that was described previously in Deliverable D4.2d [LNH+16]. At the end of Year 3, all tools support traceability by sending messages in a standardized format to the daemon, which stores the information in a database. Furthermore, users can easily query this database to retrieve specific information about the links between different entities, such as requirements, users, test results or models (FMUs). These pre-defined queries are seamlessly integrated in the INTO-CPS application. This document is closely related to Deliverables D3.1b [FGPP15], D3.2b [FGPP16] and D3.3b [FGP17b], where the foundations for traceability in INTO-CPS are described.

Contents

1 Introduction

This deliverable presents the implementation of the traceability and model management functions in INTO-CPS at the end of Year 3. It is to a large extent connected with and based on the foundational work presented in Deliverables D3.1b [FGPP15], D3.2b [FGPP16] and D3.3b [FGP17b], and on the traceability design described in Deliverable D4.2d [LNH+16]. After the introduction (Section 1), Section 2 describes the traceability actions that are recorded by the different tools. These have been extended and improved during Year 3, such that the whole tool-chain now generates traceability data in the same format. All the steps of the workflow (as described in Deliverable D3.3a [FGP17a]) are now covered by the tools. Implementation of the queries to the traceability database, which is the user-relevant result, is discussed in Section 3. This Deliverable ends with a conclusion of the status of the traceability support in the tools at the end of Year 3 and an outlook for the work that can be considered in the future (section 4).

As indicated in Deliverable D4.2d [LNH+16], it should be noted that the work presented in this Deliverable focuses on traceability. Model management, which comprises the handling of versioning of different iterations of artefacts such as models, and collaboration within larger teams, is handled by a versioning system such as Git[1]. All the tools that are described in this Deliverable rely on Git.

All the requirements that are related to traceability (summarized in Section 2.1 of Deliverable D4.2d [LNH+16] and fully documented in Deliverable D7.7 [LPO+17]) are fulfilled.

2 Implementation in the tools

The traceability architecture is described in Deliverable D4.2d [LNH+16], the traceability actions are discussed in Deliverable D3.2b [FGPP16]. To avoid redundancy, the details are therefore not repeated here.

In summary, it can be said that all of the INTO-CPS tools support traceability according to the traceability ontology, laid out in Deliverable D3.2b [FGPP16], without requiring any user interaction. The tools all follow the same message format (see Annex B) and thereby support traceability throughout the complete tool-chain. Only the IP address and port for the daemon

[1] see `https://git-scm.com/`

need to be set once in the tools, and the user name and e-mail address are defined. The traces themselves are either sent automatically during the saving of a model, or after the execution of an action. Configuration and usage of the tools is described in the User manual, Deliverable D4.3a [BLL+17].

2.1 Schema for the traceability messages

During Year 3, a schema for the traceability messages was developed, which defines the format of the messages, and restricts their content. The schema acts as an interface between the daemon and the tools. The tools need to implement the correct format, and the daemon validates if the tool has done it properly. This makes sure that the database contains only information that follows the same format, and therefore can be queried easily. Crucially, since the schema is machine-readable, the validation is done automatically. Furthermore, since the schema is public, traceability can be easily implemented in other tools (e.g. tools from vendors outside the INTO-CPS consortium, for instance from the INTO-CPS association) so that these can send valid traceability messages to the daemon. In the schema, all allowed traceability-related entities, such as activities, artefacts or tools are contained. Relations between entities, such as *prov:wasGeneratedBy*, *oslc:satisfies* and more are also described in the schema. It is therefore important in such a tool-chain-wide approach as with traceability, that all the tools comply with the schema and that the whole ontology (see Deliverable D3.3b [FGP17b]) is covered by it. The schema version 1.5 is shown in the Annex B.

The process of generating messages, sending them to the daemon and validating them, is shown in the Figure 1 below, including an example message from Modelio. In the tool, the information about the user (the *prov:Agent*) is set. The tool (here *Modelio* of type *Architecture Tool*) generates the relation (*prov:wasAssociatedWith* and *prov:used*) between the ModelDescription.xml file, the user who generated it, and the original SysML model. Then, this message is sent via HTTP to the daemon, who validates the message according to the schema.

In the remainder of this section, implementation of traceability and generation of the related messages in the different tools is described.

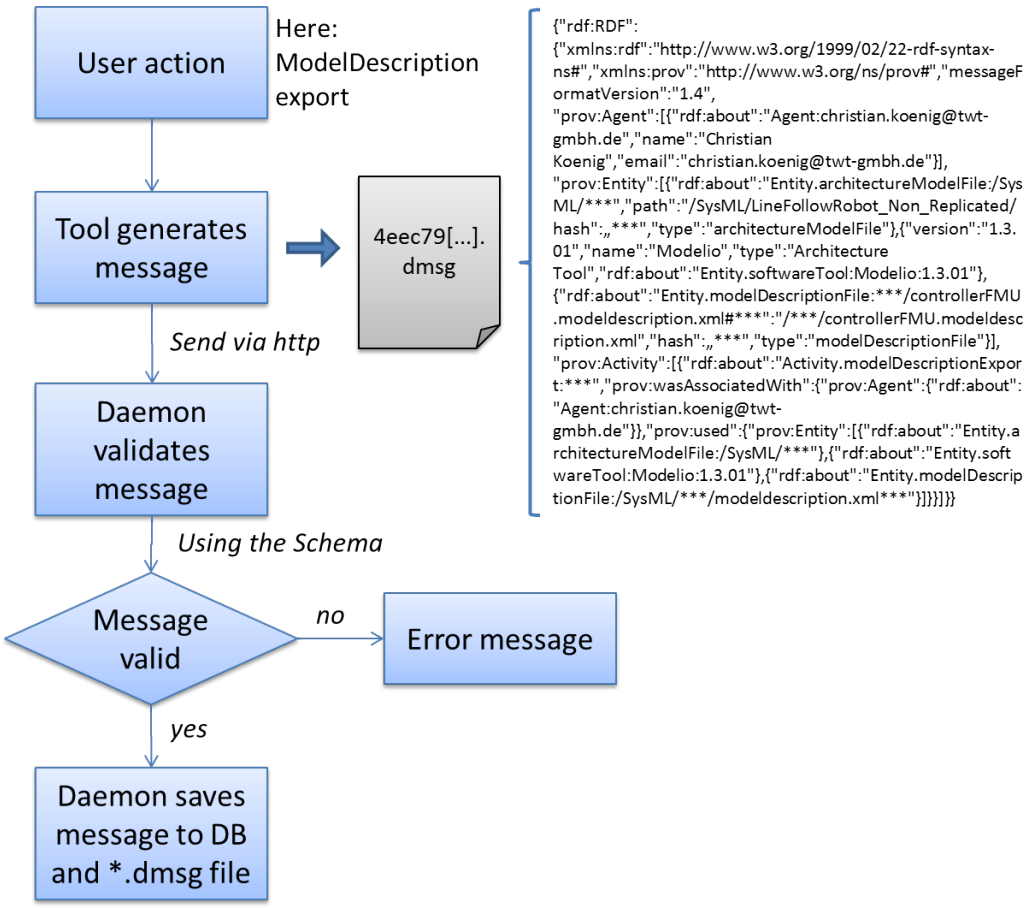

Figure 1: Schematic process of generating and saving a traceability message. For readability, the message is shortened.

2.2 Traceability Daemon

The core of the traceability architecture is the *daemon*, which receives the data from the different tools and writes it into the Neo4J database. The daemon was previously described in Deliverable D4.2d [LNH+16]. The database is stored in a binary format, which causes problems when it is versioned (e.g. in a Git or SVN system) and changed by multiple users simultaneously. To solve this, the daemon was improved in Year 3, by adding a step between receiving of the traceability messages and storing them in the Neo4J database.

Now, each message the daemon receives is saved in plain text into a single file (with .dmsg file ending) in the project folder. The content of one such file is

indicated in Figure 1. At startup of the INTO-CPS Application, the daemon builds the database from these single files. This allows multiple users to work on the project simultaneously. Each user generates traceability messages by the different actions he/she performs. These messages are stored in the project folder. After completion of a task, the user pushes the files to the repository. Merging of the database is then done by Git / SVN by combining the .dmsg files. After an update of the project folder, each user has access to the whole database. The schematic process is shown in Figure 2 below.

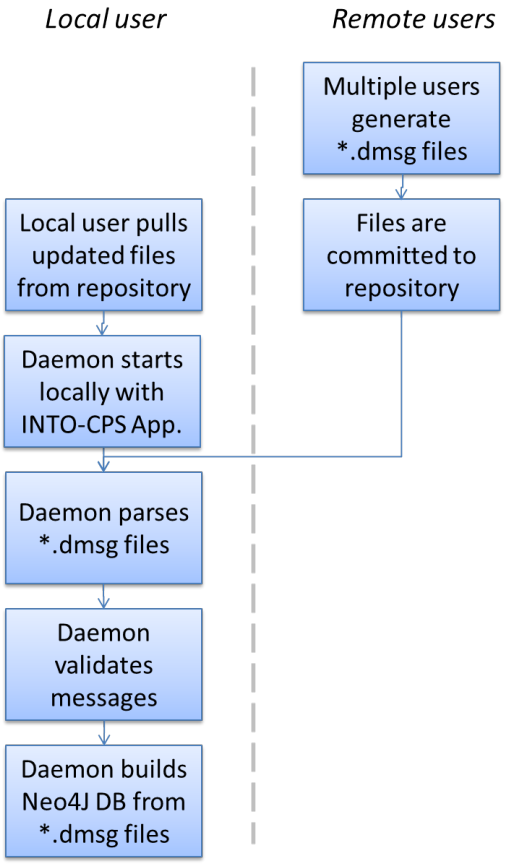

Figure 2: Schematic process of merging multipe messages from different users and building the Neo4J database from them.

In addition, the daemon is validating the messages it receives from the tools with the schema (see Annex B), to make sure that only those messages that comply with the schema are written into the database. Only when all tools use the same message format will the queries (see Section 3) return meaningful information.

2.3 Modelio

Modelio records the following traceability actions:

- Architecture creation
- Architecture modification
- (ModelDescription import)
- ModelDescription export
- Co-Simulation configuration export
- Requirements generation and linking to SysML blocks

Modelio represents the *Architecture Modeling* activity in the INTO-CPS workflows (see Section 3 in Deliverable D3.3a [FGP17a]). Consequently, these actions are traced[2]. The Architecture creation / modification captures the generation and modification of a SysML block in Modelio. The generation of ModelDescription.xml files from a SysML block is the next step in the workflow. Generation of SysML blocks from imported ModelDescription.xml files is not yet traced, but planned for the future (and will most likely be carried out in the INTO-CPS association, see Deliverable D6.3 [KFP+17]). Exporting a co-simulation configuration from a SysML connections diagram, which can transformed in the INTO-CPS application into a Multi-model, is also traced. Generation of requirements, and association of these requirements with SysML blocks is traced. This association can either be of the type *verify* or *satisfy*.

In addition to the ad-hoc generation of traceability messages, which are created when the related action is being performed, Modelio also offers the option to convert the Git history of a Modelio project into traceability messages. This is particularly useful for use-cases, where traceability was not used from the very beginning.

2.4 Modeling tools

Since OpenModelica, 20-sim and Overture are modeling tools, they are here described together. Generation of models, either from scratch or from an imported ModelDescription.xml file (e.g. coming from Modelio in the previous

[2]In the context of this deliverable, "traced" means that messages are generated and sent to the daemon

step) is the next step in the workflow, and consequently traced, together with their modification. FMUs can be imported from other tools, to include them in the native models. Exporting an FMU is the next step in the workflow, and is consequently also traced. OpenModelica, 20-sim and Overture record the following traceability actions:

- Model creation
- Model modification
- FMU export
- FMU import
- ModelDescription import

2.4.1 20-sim

Configuration of the 20-sim traceability support is described in the User manual, Deliverable D4.3a [BLL+17]. Therefore the steps for configuration are not repeated here in detail.

Because Git is needed for the INTO-CPS traceability daemon, it is not possible to only enable the traceability daemon without enabling Git version control. If both options (for Git version control and for communication with the traceability daemon) are enabled, every traceable action in 20-sim will store a copy of its data in the indicated Git repository. If the model itself is already in a Git repository, this will also make sure to commit the changes to this repository automatically. There is an additional option named "Write custom save messages", which will ask the user to write a custom message whenever a traceable action is performed. This message will be stored in Git as the Git commit message.

The "Model creation" action is a "Save as" action, which is the moment when the user officially saves a new model to disk. In the same line of reasoning, a "Model modification" action is a "Save" action in 20-sim, because the user modifies an existing model on disk. 20-sim has no support for deleting a model from within its user interface, therefore there is no traceability query to delete a model from 20-sim. 20-sim also has support to export and import an FMU and to import a modelDescription.xml file. These three actions are described in [BLL+17], and are also traceable. The exported or imported FMU or the imported modelDescription.xml file will also be placed under

version control in the Git repository. Currently 20-sim does not support tracing the export of a tool-wrapper FMU.

2.4.2 OpenModelica

Traceability support in OpenModelica is very similar to the one implemented in 20-sim. After an initial configuration of the Git repository and and traceability daemon, the actions for saving a model, import of a ModelDescription.xml file and export of a FMU are traced without further user interaction.

2.4.3 Overture

In Overture, traceability is implemented as an additional package (as a `.jar` file), that can be downloaded from the GitHub page [3]. This package extracts traceability information from the Git repository, where the current Overture project is stored in. It can be either triggered manually, or simply added to a Git post-commit hook, to send new traces to the daemon after the user commits the changes to the model to the repository. Similar to Modelio, this way of extracting traceability messages from the Git repository is useful if traceability has not been used since the start of the project.

2.5 RT Tester

RT Tester records the following traceability actions:

- Define test model
- Define test objectives
- Run test
- Define model-checking model
- Define continuous time abstraction
- Run model-checking query

There is no need for the user to configure these operations, because per default valid settings (for the INTO-CPS Application) are used.

[3] see `https://github.com/overturetool/intocps-tracability-driver/releases`

The following environment variables can be used to modify the tracing behaviour.

`OSLC_SERVER`	machine running the OSLC server (default: localhost)
`OSLC_PORT`	port number to address the OSLC server at (default: 8083)
`OSLC_EDOMAIN`	email-domain to use (default: example.com)
`OSLC_VERBOSE`	if set (to non-0/non-False), then generate verbose output that shows all json data before posting

Example. In a project, a test "TR-TR" has been configured that aims to cover all transition relations. By this computation, the requirements REQ-001 and REQ-002 have been covered (by one or more model elements), but the test run did not reach the model element that is related to REQ-004.

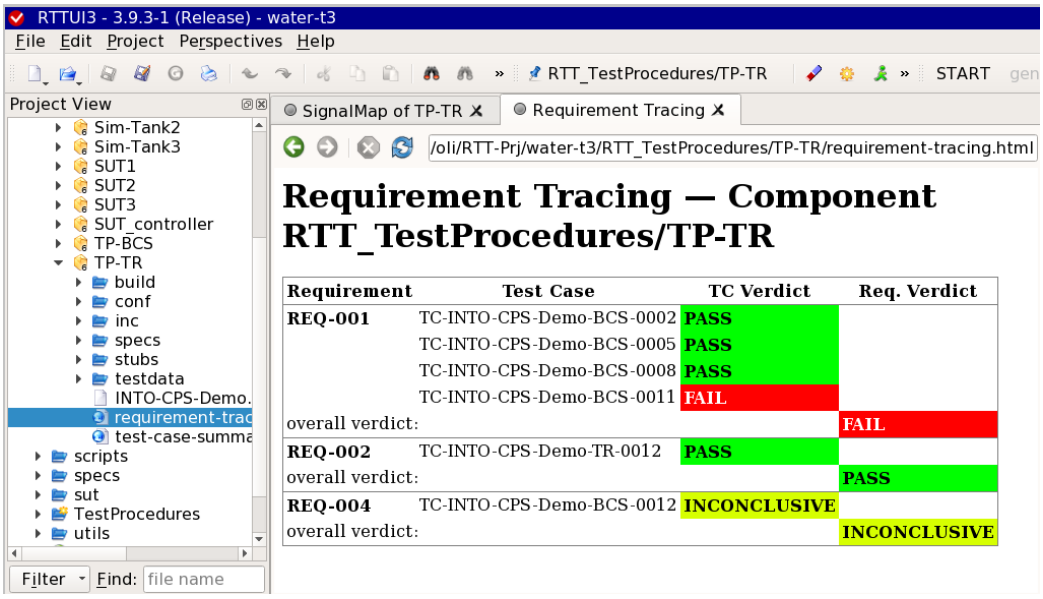

Figure 3: RT-Tester Test-Automation result, requirements may be PASS, FAIL, or INCONCLUSIVE (i.e., untried).

After the "Run test" operation, the test result is shown in Figure 3: The REQ-001 is marked FAIL (because one of the reached control states did not behave as expected), REQ-002 is PASS, and REQ-004 is INCONCLUSIVE, i.e., it is unclear whether the system under test would behave correctly here - the situation has not been reached.

This information is transmitted (automatically) to the Neo4J server, along with related information like what tool/what version has been used, who

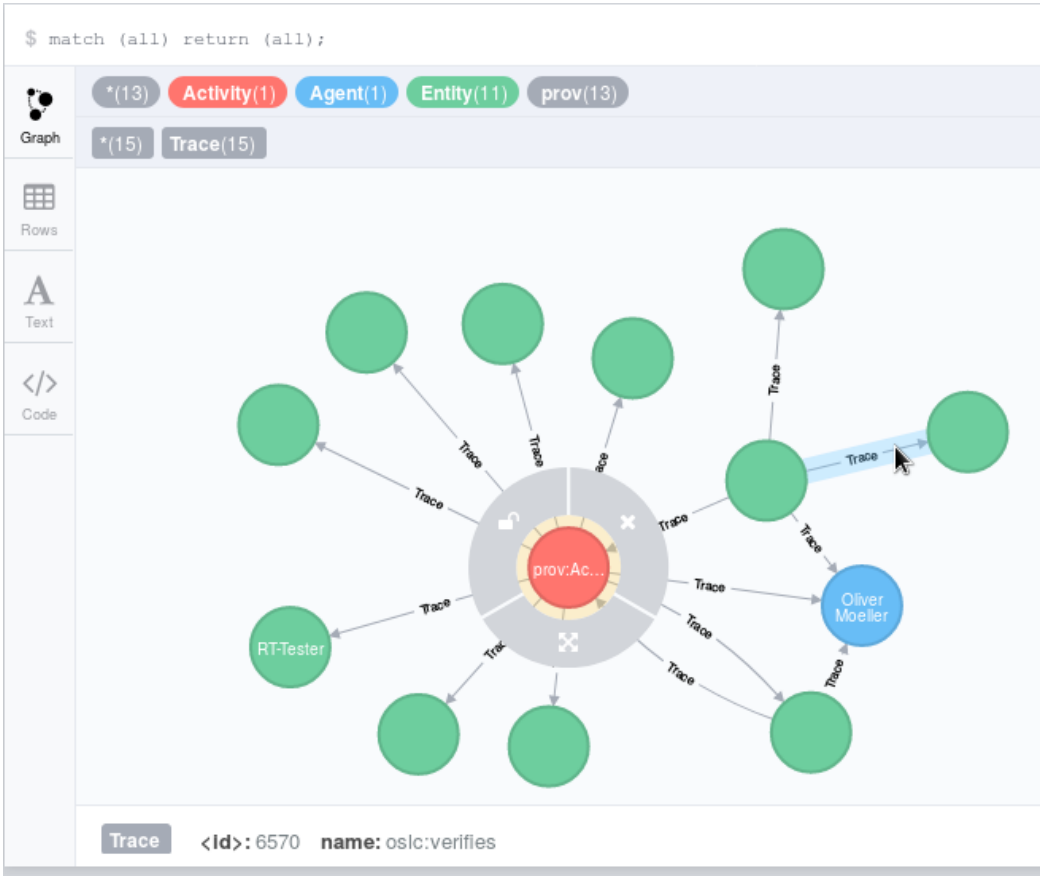

Figure 4: Representation of the "Run test" activity in the Neo4J database.

performed the operation, etc. This is shown in Figure 4: The highlighted arrow points to REQ-002 (which is PASS and thus **verified**). The other "Trace" relation with the same origin points to REQ-002 and has the name "into:violates", because the test run demonstrates that REQ-001 does not (always) hold. The REQ-004 is not connected to the test run (and would be a separate green blob here, not displayed).

2.6 INTO-CPS Application

The INTO-CPS Application records the following traceability actions:

- Multi-model creation
- Co-Simulation configuration creation

- Run Simulation

These actions are automatically recorded once the user creates a multi-model from an exported SysML configuration diagram, generates an Co-Simulation configuration from a multi-model, or modifies these configurations. Finally, the start of a simulation run is also recorded.

3 Querying and Visualisation

In order to bring a benefit to the user, the traceability data not only needs to be recorded, but also analysed and presented in a way that is helpful to the user. The tools therefore must have a way of querying the database, for specific information, such as relations between requirements, models, test results, users or simulation results.

The results from these queries are displayed within the INTO-CPS Application as lists, separated between different categories (FMUs, Users, Simulations, Requirements), as discussed below in Section 3.2. These categories can be extended and minimized, to present a neatly arranged view to the user. The interface has the same look as the rest of the INTO-CPS Application, which makes it user-friendly. Additionally, for expert users that have a good understanding of the underlying structure, and that are proficient in generating queries to the database, it is possible to manually enter queries to search the traceability database. This is briefly described in Section 3.3.

While there is plenty of research on traceability in software or systems engineering, only few industry standard tools implement traceability. One of them, IBM Doors Next Generation, is among the most popular tools [WP10], which displays traceability relations between requirements on different levels (e.g. high-level requirements and their refinements) as trees or lists [4]. Another popular way of displaying traceability relations is the matrix view, which shows the relations between different artefacts in a 2-dimensional table. However, due to the heterogeneity of the different artefact types (requirements, models / FMUs, simulation results, configuration files etc.), the matrix view is not implemented in the context of INTO-CPS. Another popular way of presenting links is the graph view, where the different artefacts and their relation is shown in a graph. This is possible using the expert mode, which is based on the Neo4J interface, as described below in Section

[4] see also `https://jazz.net/library/article/88104`

3.3. In principle however, the openness of the INTO-CPS tool-chain allows for creation of new views, if they are required by a specific use-case.

3.1 Cypher query language

The Neo4J database uses a query language called *Cypher*. This language uses ASCII art to represent nodes and relations. Nodes are surrounded by parentheses "(" and ")", and relationships are identified by square brackets "[" and "]". More information can be found on `https://neo4j.com/developer/cypher-query-language/`.

3.2 Implementation in the INTO-CPS application

For representation of the traceability links to the users, pre-defined queries were integrated to the INTO-CPS Application. They allow the user to search for different artefacts and relations between artefacts. The user interface is identical to the rest of the INTO-CPS Application, which lowers the entry barriers for users. The searches generate lists of items, which can be minimized to keep an overview of all the presented data.

At the end of Year 3, the following queries (which are also described in Annex B of Deliverable D3.3b [FGP17b]) are implemented in the INTO-CPS Application to allow for a easy usage.

1. FMUs: Query the database for all requirements that are related to a specific FMU.
2. Users: Query the database for all activities and artefacts that are related to a specific user.
3. Simulations: Query the database for all the Co-Simulation results that are associated with a multi-model.
4. Requirements: Query the database for test results that are linked to requirements.

Right-clicking on the "Traceability" button on the left-hand side of the window opens a context menu (see Figure 5). Clicking on "Trace Objects" shows the overview of the different queries, that can then be extended and minimized.

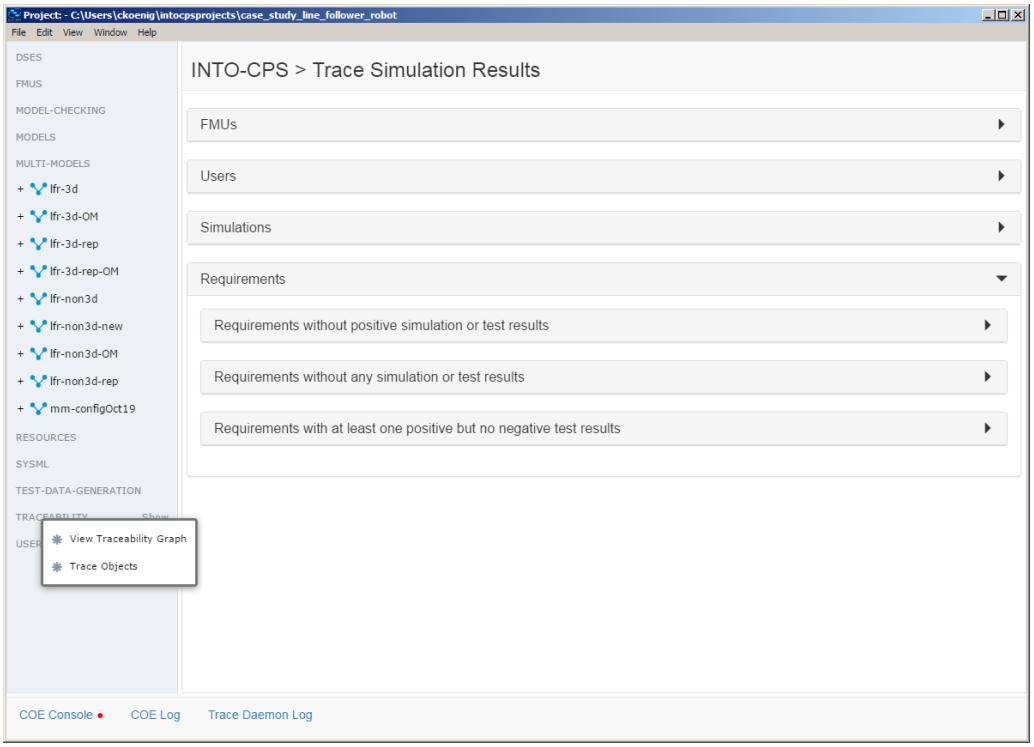

Figure 5: Overview of the traceability queries in the INTO-CPS Application.

FMU and requirements This query is done in two steps. The first query lists all the FMUs that are stored in the database (e.g. after FMU export in Overture, 20-sim or OpenModelica, see Section 2.4.)

In the Cypher language (see previous Section), the first query is done with the following command:

```
match(n{type:'fmu'}) return n.uri, n.path
```

In the next step, all requirements that are related to a specific FMU (`<FMU_name>`) are queried by this command:

```
match (activity)<-[:Trace{name:"prov:wasGeneratedBy"}]-
({uri:'<FMU_name>'})-[:Trace{name:"oslc:satisfies"}]->(element)
return element.uri, element.hash, activity.time, element.type
order by activity.time desc
```

This returns all the requirements that are linked by the *oslc:satisfies* relationship to the particular FMU.

Users and artefacts and activities The INTO-CPS project aims explicitly the collaborative modelling, which means that multiple people are typically involved in the process. To support this, all the users and their actions can be traced.

First, all users are queried from the database by the following command:

```
match (usr{specifier:'prov:Agent'}) return usr.name, usr.uri
```

Next, all the artefacts that were influenced by a particular user (here identified by the URI, which contains the e-mail address *Agent.richard.payne@newcastle.ac.uk*) can be found by:

```
match (usr{uri:'Agent.richard.payne@newcastle.ac.uk'})<-
[:Trace{name:'prov:wasAttributedTo'}]-(entity)
return entity.uri, entity.type
```

These artefacts are for example simulation results, FMUs, model description files or simulation configurations. A complete list of activites can be found in Annex B under the enumeration for `ArtefactType`.

In addition, all the activities performed by this user can be traced by:

```
match (usr{uri:'Agent.richard.payne@newcastle.ac.uk'})<-
[:Trace{name:'prov:wasAssociatedWith'}]-(entity)
return entity.uri, entity.type
```

The activites are for example architectureModelling, modelDescriptionExport, simulationModelling and so forth. A complete list of activites can be found in Annex B under the enumeration for `ActivityType`. Those activites reflect the activites as described in the ontology (see also Annex A of Deliverable D3.3b [FGP17b]).

Simulation results and files First all the simulation results are queried by the following command:

```
match (n{type:'simulationResult'})-
[:Trace{name:"prov:wasGeneratedBy"}]->(m)
return n.uri, m.time, m.type
```

In the next step, all files that were used (i.e. that have the relation *prov:used*) to produce this simulation result (`<Result_file>`) are queried by the following command:

```
match({uri:'Entity.<Result_file>'})-
```

```
[:Trace{name:"prov:wasGeneratedBy"}]->
(simulation)-[:Trace{name:"prov:used"}]-(entity)
return entity.uri, entity.path, entity.hash
```

This query lists the FMUs, the configuration files and the log files which are related to this particular simulation result.

Requirements and Test results To take the test results from RT Tester into account (see Section 2.5), three different queries were implemented.

Requirements without positive simulation or test results are queried by:

```
match (req{type:'requirement'}) where not
(req)<-[:Trace{name:"oslc:verifies"}]-()
return req.uri
```

This query indicates to the user all those requirements that have not been validated yet.

Requirements without any simulation or test result are queried by:

```
match (req{type:'requirement'}) where not (req)<-
[:Trace{name:"into:violates"}]-() and not (req)<-
[:Trace{name:"oslc:verifies"}]-()
return req.uri
```

This query finds all requirements that have not yet been tested, to indicate to the user which

And finally, requirements with at least one positive but no negative test result are queried by:

```
match (req{type:'requirement'}) where (req)<-
[:Trace{name:"oslc:verifies"}]-() and not
(req)<-[:Trace{name:"into:violates"}]-()
return req.uri
```

This finds those requirements that have been tested positively and can be seen as fulfilled, since no counter-example was found.

3.3 Expert User Queries

In addition to these built-in queries, expert queries can be performed by the native Neo4J interface that is integrated in the INTO-CPS application, using

the Cypher language (see Section 3.1). To access this interface, the user can click on "View Traceability Graph" in the context menu for traceability (see Figure 5). The interface, with the graph view, is shown in Figure 4. The queries, for example those described in the previous Section, can be directly typed into the command line, returning lists of objects. Advanced users can modify the queries from the previous sections, use those described in Annex B of Deliverable D3.3b [FGP17b] or define their own queries.

In the framework of the INTO-CPS project, visualisation of the query results is focused to the INTO-CPS application. The traceability architecture allows however in principle for querying of the database from any other tool. For example, OpenModelica also implemented viewing the graph database within OpenModelica.

The usage of the queries in the INTO-CPS application is described in Deliverable D4.3a [BLL+17] and is therefore not repeated here.

4 Summary

This deliverable presents the status of the traceability and model management efforts in INTO-CPS at the end of Year 3. Continuing the work from the previous years, a message schema was defined that ensures that all tools use the same format for sending their data. The handling of the Neo4J database by the daemon was improved to allow working with multiple users on a repository. All tools record the relevant actions, and the whole workflow of INTO-CPS is covered, with respect to traceability data. Queries were implemented in the INTO-CPS application to return meaningful data to the user.

While the INTO-CPS tool-chain is well covered with respect to traceability by the end of Year 3, external tools are not supported. For example, if FMUs were generated in other tools, this is not listed in the traceability database. Therefore, methods for covering these artefacts coming from external tools, could be developed in the future. Since the interface, the ontology and the format for the messages are public, however, support for external tools can easily be integrated by their developers. In principle, traceability should be used since the beginning of a project, such as CPS design. However, parsing of the Git repository, as it is enabled by Overture or Modelio, enables users to take advantage of traceability even though it was not used from the very beginning.

There is a plethora of research on traceability in software and systems engineering. In the context of INTO-CPS, we enabled traceability in the whole tool-chain of CPS design, from systems modelling, trough physical and cyber modeling, down to co-simulation and test automation. This presents an important step in the true integration of the different tools that are used in CPS design.

Appendices

A Abbreviations

ASCII	American Standard for Information Interchange
COE	Co-Simulation Orchestration Engine
CPS	Cyber-Physical System
DB	Database
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
HiL	Hardware in the Loop
JSON	JavaScript Object Notation
OSLC	Open Services for Lifecycle Collaboration
SVN	Apache Subversion
URI	Uniform Resource Identifier
XML	eXtensible Markup Language

B Traceability Schema v1.5

In the following, the schema for sending traceability messages is listed. It ensures that all tools send the messages in the same format, so that they can be queried from the database later on.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "INTO-CPS Traceability JSON Schema",
  "version": "1.5",
  "type": "object",
  "properties": {
    "rdf:RDF": {
      "type": "object",
      "minProperties": 4,
      "maxProperties": 6,
      "properties": {
        "xmlns:rdf": {
          "type": "string",
          "enum": [
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          ],
          "default": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        },
        "xmlns:prov": {
          "type": "string",
          "enum": [
            "http://www.w3.org/ns/prov#"
          ],
          "default": "http://www.w3.org/ns/prov#"
        },
        "messageFormatVersion": {
          "type": "string",
          "enum": [
            "1.3",
            "1.3.1",
            "1.3.2",
```

```
            "1.4",
            "1.5"
          ],
          "default": "1.5"
        },
        "prov:Agent": {
          "type": "array",
          "minItems": 1,
          "uniqueItems": true,
          "items": [
            {
              "$ref": "#/definitions/Agent"
            }
          ],
          "additionalItems": {
            "$ref": "#/definitions/Agent"
          }
        },
        "prov:Entity": {
          "type": "array",
          "minItems": 1,
          "uniqueItems": true,
          "items": {
            "anyOf": [
              {
                "$ref": "#/definitions/Artefact"
              },
              {
                "$ref": "#/definitions/Tool"
              }
            ]
          }
        },
        "prov:Activity": {
          "type": "array",
          "minItems": 1,
          "uniqueItems": true,
          "items": [
            {
              "$ref": "#/definitions/Activity"
            }
```

```
          ],
          "additionalItems": {
            "$ref": "#/definitions/Activity"
          }
        }
      },
      "required": [
        "xmlns:rdf",
        "xmlns:prov",
        "messageFormatVersion"
      ],
      "additionalProperties": false
    }
  },
  "required": [
    "rdf:RDF"
  ],
  "additionalProperties": false,
  "definitions": {
    "ActivityType": {
      "type": "string",
      "enum": [
        "architectureConfigurationCreation",
        "architectureModelling",
        "codeGeneration",
        "configurationCreation",
        "defineCTAbstraction",
        "defineMCModel",
        "defineMCQuery",
        "defineTestModel",
        "defineTestObjectives",
        "designNoteCreation",
        "dse",
        "dseAnalysisCreation",
        "dseConfigurationCreation",
        "fmuExport",
        "fmuImport",
        "fmuExportForHiL",
        "mockupFMUCreation",
        "modelChecking",
        "modelCreation",
```

```
114           "modelModification",
115           "modelDeletion",
116           "modelDescriptionExport",
117           "modelDescriptionImport",
118           "modelPortionFMUExport",
119           "requirementsManagement",
120           "runMCQuery",
121           "runTest",
122           "simulation",
123           "simulationConfigurationCreation",
124           "simulationModelling",
125           "testCreation"
126         ]
127       },
128       "ArtefactType": {
129         "type": "string",
130         "enum": [
131           "architectureConfiguration",
132           "architectureConnectionDiagram",
133           "architectureModelFile",
134           "architectureStructureDiagram",
135           "architectureSubSystem",
136           "dseAlgorithm",
137           "dseAnalysisScript",
138           "dseRankingScript",
139           "dseRankingValue",
140           "dseResult",
141           "dseSearchConfiguration",
142           "dseAnalysisConfiguration",
143           "designNote",
144           "designNoteFile",
145           "fmu",
146           "file",
147           "hiLAsset",
148           "modelCheckingAbstraction",
149           "modelCheckingQuery",
150           "modelCheckingResult",
151           "modelCheckModel",
152           "modelCheckResult",
153           "modelDescriptionFile",
154           "modelFile",
```

```
155           "modelPortionConfiguration",
156           "multiModelConfiguration",
157           "objectivesValue",
158           "requirement",
159           "requirementSource",
160           "requirementSourceSubPart",
161           "requirementsDocument",
162           "scenarioData",
163           "simulationConfiguration",
164           "simulationModelContainer",
165           "simulationResult",
166           "softwareAgent",
167           "testCase",
168           "testConfiguration",
169           "testExecutionResult"
170         ]
171       },
172       "ToolType": {
173         "type": "string",
174         "enum": [
175           "Architecture Tool",
176           "Co Simulation Engine",
177           "Co Simulation GUI",
178           "Software Tool",
179           "Simulation Tool",
180           "Model Checking Tool",
181           "Test Automation Tool"
182         ]
183       },
184       "Activity": {
185         "type": "object",
186         "minProperties": 5,
187         "maxProperties": 6,
188         "properties": {
189           "rdf:about": {
190             "$ref": "#/definitions/URIActivity"
191           },
192           "type": {
193             "$ref": "#/definitions/ActivityType"
194           },
195           "time": {
```

```
196             "type": "string",
197             "format": "date-time"
198           },
199           "prov:wasAssociatedWith": {
200             "type": "object",
201             "minProperties": 1,
202             "maxProperties": 1,
203             "properties": {
204               "prov:Agent": {
205                 "$ref": "#/definitions/RefAgent"
206               }
207             },
208             "required": [
209               "prov:Agent"
210             ],
211             "additionalProperties": false
212           },
213           "prov:used": {
214             "type": "object",
215             "minProperties": 1,
216             "maxProperties": 1,
217             "properties": {
218               "prov:Entity": {
219                 "type": "array",
220                 "minItems": 1,
221                 "uniqueItems": true,
222                 "items": {
223                   "anyOf": [
224                     {
225                       "$ref": "#/definitions/RefTool"
226                     },
227                     {
228                       "$ref": "#/definitions/
                            RefArtefact"
229                     }
230                   ]
231                 }
232               }
233             },
234             "required": [
235               "prov:Entity"
```

```
          ],
          "additionalProperties": false
        }
      },
      "required": [
        "rdf:about",
        "type",
        "time",
        "prov:wasAssociatedWith",
        "prov:used"
      ],
      "additionalProperties": false
    },
    "URIArtefact": {
      "type": "string",
      "pattern": "^Entity\\.(
        architectureConfiguration|
        architectureConnectionDiagram|
        architectureModelFile|
        architectureStructureDiagram|
        architectureSubSystem|dseAlgorithm|
        dseAnalysisScript|dseRankingScript|
        dseRankingValue|dseResult|
        dseSearchConfiguration|
        dseAnalysisConfiguration|designNote|
        designNoteFile|fmu|file|hiLAsset|
        modelCheckingAbstraction|modelCheckingQuery
        |modelCheckingResult|modelCheckModel|
        modelCheckResult|modelDescriptionFile|
        modelFile|modelPortionConfiguration|
        multiModelConfiguration|objectivesValue|
        requirement|requirementSource|
        requirementSourceSubPart|
        requirementsDocument|scenarioData|
        simulationConfiguration|
        simulationModelContainer|simulationResult|
        softwareAgent|testCase|testConfiguration|
        testExecutionResult):([a-zA-Z0-9\\/.\\-_])
        +(:([a-zA-Z0-9\\.-_])+)?#([0-9a-f]{5,40}|[0
        ])$",
      "additionalProperties": false
```

```
253     },
254     "URITool": {
255       "type": "string",
256       "pattern": "^Entity\\.(architectureTool|
            coSimulationEngine|coSimulationGUI|
            softwareTool|simulationTool|
            modelCheckingTool|testAutomationTool):([a-
            zA-Z0-9\\/.\\-_]+)(:[a-zA-Z0-9\\/.\\-_ ()]
            +)*$",
257       "additionalProperties": false
258     },
259     "URIActivity": {
260       "type": "string",
261       "pattern": "^Activity\\.(
            architectureConfigurationCreation|
            architectureModelling|codeGeneration|
            configurationCreation|designNoteCreation|
            dse|dseAnalysisCreation|
            dseConfigurationCreation|fmuExport|
            fmuImport|fmuExportForHiL|mockupFMUCreation
            |modelCreation|modelModification|
            modelDeletion|modelDescriptionExport|
            modelChecking|modelDescriptionImport|
            modelPortionFMUExport|
            requirementsManagement|simulation|
            simulationConfigurationCreation|
            simulationModelling|testCreation|
            defineTestModel|defineTestObjectives|
            runTest|defineMCModel|defineCTAbstraction|
            defineMCQuery|runMCQuery):([0-9]){4}\\-([0-
            1][0-9])\\-([0-3][0-9])T([0-2][0-9]):([0-5]
            [0-9]):([0-5][0-9])(\\.[0-9][0-9][0-9])?Z#[
            a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}
            -[a-fA-F0-9]{4}-[a-fA-F0-9]{12}$",
262       "additionalProperties": false
263     },
264     "URIAgent": {
265       "type": "string",
266       "pattern": "^Agent\\:[a-z0-9._-]+@[a-z0-9._-]{
            2,}\\.[a-z]{2,4}$",
267       "additionalProperties": false
```

```
    },
    "RefEntities": {
      "type": "object",
      "minProperties": 1,
      "maxProperties": 1,
      "properties": {
        "prov:Entity": {
          "type": "array",
          "minItems": 1,
          "uniqueItems": true,
          "items": [
            {
              "$ref": "#/definitions/RefArtefact"
            }
          ],
          "additionalItems": {
            "$ref": "#/definitions/RefArtefact"
          }
        }
      },
      "additionalProperties": false
    },
    "RefArtefact": {
      "type": "object",
      "minProperties": 1,
      "maxProperties": 1,
      "properties": {
        "rdf:about": {
          "$ref": "#/definitions/URIArtefact"
        }
      },
      "required": [
        "rdf:about"
      ],
      "additionalProperties": false
    },
    "RefTool": {
      "type": "object",
      "minProperties": 1,
      "maxProperties": 1,
      "properties": {
```

```
309           "rdf:about": {
310             "$ref": "#/definitions/URITool"
311           }
312         },
313         "required": [
314           "rdf:about"
315         ],
316         "additionalProperties": false
317       },
318       "RefActivity": {
319         "type": "object",
320         "minProperties": 1,
321         "maxProperties": 1,
322         "properties": {
323           "rdf:about": {
324             "$ref": "#/definitions/URIActivity"
325           }
326         },
327         "required": [
328           "rdf:about"
329         ],
330         "additionalProperties": false
331       },
332       "RefAgent": {
333         "type": "object",
334         "minProperties": 1,
335         "maxProperties": 1,
336         "properties": {
337           "rdf:about": {
338             "$ref": "#/definitions/URIAgent"
339           }
340         },
341         "required": [
342           "rdf:about"
343         ],
344         "additionalProperties": false
345       },
346       "Agent": {
347         "type": "object",
348         "minProperties": 2,
349         "maxProperties": 3,
```

```
      "properties": {
        "rdf:about": {
          "$ref": "#/definitions/URIAgent"
        },
        "name": {
          "type": "string"
        },
        "email": {
          "type": "string",
          "format": "email"
        }
      },
      "required": [
        "rdf:about",
        "email"
      ],
      "additionalProperties": false
    },
    "Tool": {
      "type": "object",
      "properties": {
        "version": {
          "type": "string"
        },
        "name": {
          "type": "string"
        },
        "type": {
          "$ref": "#/definitions/ToolType"
        },
        "rdf:about": {
          "$ref": "#/definitions/URITool"
        }
      },
      "required": [
        "version",
        "name",
        "type",
        "rdf:about"
      ],
      "additionalProperties": false
```

```
    },
    "Artefact": {
      "type": "object",
      "maxProperties": 13,
      "properties": {
        "rdf:about": {
          "$ref": "#/definitions/URIArtefact"
        },
        "path": {
          "type": "string",
          "pattern": "^([a-zA-Z0-9\\/.\\-_ ])+$"
        },
        "hash": {
          "type": "string",
          "pattern": "^([0-9a-f]{5,40}|[0])$"
        },
        "type": {
          "$ref": "#/definitions/ArtefactType"
        },
        "prov:wasAttributedTo": {
          "type": "object",
          "minProperties": 1,
          "maxProperties": 1,
          "properties": {
            "prov:Agent": {
              "$ref": "#/definitions/RefAgent"
            }
          },
          "required": [
            "prov:Agent"
          ],
          "additionalProperties": false
        },
        "prov:wasGeneratedBy": {
          "type": "object",
          "minProperties": 1,
          "maxProperties": 1,
          "properties": {
            "prov:Activity": {
              "$ref": "#/definitions/RefActivity"
            }
```

```
            },
            "required": [
              "prov:Activity"
            ],
            "additionalProperties": false
          },
          "prov:wasDerivedFrom": {
            "$ref": "#/definitions/RefEntities"
          },
          "prov:hadMember": {
            "$ref": "#/definitions/RefEntities"
          },
          "oslc:elaborates": {
            "$ref": "#/definitions/RefEntities"
          },
          "oslc:satisfies": {
            "$ref": "#/definitions/RefEntities"
          },
          "oslc:verifies": {
            "$ref": "#/definitions/RefEntities"
          },
          "into:doesNotVerify": {
            "$ref": "#/definitions/RefEntities"
          },
          "into:violates": {
            "$ref": "#/definitions/RefEntities"
          }
        },
        "required": [
          "rdf:about",
          "path",
          "hash",
          "type"
        ],
        "additionalProperties": true
      }
    }
}
```

References

[BLL+17] Victor Bandur, Peter Gorm Larsen, Kenneth Lausdahl, Casper Thule, Anders Franz Terkelsen, Carl Gamble, Adrian Pop, Etienne Brosse, Jörg Brauer, Florian Lapschies, Marcel Groothuis, Christian Kleijn, and Luis Diogo Couto. INTO-CPS Tool Chain User Manual. Technical report, INTO-CPS Deliverable, D4.3a, December 2017.

[FGP17a] John Fitzgerald, Carl Gamble, and Ken Pierce. Method Guidelines 3. Technical report, INTO-CPS Deliverable, D3.3a, December 2017.

[FGP17b] John Fitzgerald, Carl Gamble, and Ken Pierce. Methods Progress Report 3. Technical report, INTO-CPS Deliverable, D3.3b, December 2017.

[FGPP15] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Methods Progress Report 1. Technical report, INTO-CPS Deliverable, D3.1b, December 2015.

[FGPP16] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Methods Progress Report 2. Technical report, INTO-CPS Deliverable, D3.2b, December 2016.

[KFP+17] Christian König, Peter Fritzson, Adrian Pop, Christian Kleijn, Peter Gorm Larsen, Mette Stig Hansen, Jörg Brauer, and Stylianos Basagiannis. Dissemination and Exploitation Report - Year 3. Technical report, INTO-CPS Deliverable, D6.3, December 2017.

[LNH+16] Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble, Oliver Mölle, Etienne Brosse, Tom Bokhove, Luis Diogo Couto, and Adrian Pop. INTO-CPS Traceability Design. Technical report, INTO-CPS Deliverable, D4.2d, December 2016.

[LPO+17] Peter Gorm Larsen, Ken Pierce, Julien Ouy, Kenneth Lausdahl, Marcel Groothuis, Adrian Pop, Miran Hasanagic, Jörg Brauer, Etienne Brosse, Carl Gamble, Simon Foster, and Jim Woodcock. Requirements Report Year 3. Technical report, INTO-CPS Deliverable, D7.7, December 2017.

[WP10] Stefan Winkler and Jens Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling (SoSyM)*, 9(4):529–565, 2010.