Traceability in the Model-based Design of Cyber-Physical Systems

Christian König[1] Alachew Mengist[2] Carl Gamble[3] Jos Höll[1] Kenneth Lausdahl[4] Tom Bokhove[5] Etienne Brosse[6] Oliver Möller[7] Adrian Pop[2]

[1]TWT GmbH Science & Innovation, Stuttgart, Germany, christian.koenig@twt-gmbh.de
[2]Department of Computer and Information Science, Linköping University, Sweden, alachew.mengist@liu.se
[3]School of Computing, Newcastle University, Newcastle upon Tyne, United Kingdom
[4]Department of Engineering - Aarhus University, Finlandsgade 22, Aarhus N,Denmark
[5]Controllab Products, Enschede, The Netherlands
[6]Softeam Research & Development Division, Paris, France
[7]Verified Systems International Bremen, Germany

Abstract

Design, development, and analysis of complex Cyber Physical Systems (CPSs) using models involves a collaboration of expertise from different engineering domains. Heterogeneous artefacts are generated, often using different lifecycle modeling languages and simulation tools. Capturing the traceability information among these artefacts can be used to support several activities such as requirements tracing, impact analysis of change requests, verification, validation, and documentation. However, creating trace links among these heterogeneous artefacts is challenging as different tools in the development lifecycle are usually disparate and there is no precise semantic in the terminology used between requirement engineers, verification engineers, and system modelers. In this paper, we present a linked data-based approach to capture traceability information and create trace links that relate heterogeneous artefacts in the model-based design process of CPSs through a standardized interface and format using OSLC. This enables artefacts from different tools to be connected and queried through a standardized interface and format. A practical prototype system for supporting traceability is designed through integration with the INTO-CPS toolchain of CPS design. The traceability data is stored in Neo4j graph database which can be queried for generating various reports such as impact analysis, variant handling, etc.

Keywords: Traceability, Trace links, Linked data, Tool integration, OSLC, Open Service for Lifecycle Collaboration, Model Based Design, Cyber-Physical-Systems

1 Introduction

Cyber-Physical Systems (CPSs) are complex systems that typically consist of computing elements, physical elements and communication channels (Song et al., 2016). CPSs can for example be found in the aerospace domain, in automotive engineering, building automation or robotics, where they often have safety-relevant features. Therefore, exhaustive testing of the systems is necessary. To minimize cost and development time, much of the testing needs to be done virtually, while the actual device is not yet fully ready. To allow this in an efficient manner, model-based systems engineering (MBSE) methods are being applied to the design of CPSs, to develop and test those systems in a time and cost-efficient manner. The MBSE approach requires that parts and features of the system can be related to the design requirements in an automated fashion, supported by appropriate tools. This is the requirements traceability challenge, where the different steps of the implementation and validation of requirements need to be traceable to the original requirements. Only then is it possible to reliably demonstrate that all requirements have been taken into account in the design of the CPS. The software tools that are used to develop and test such complex systems are often heterogeneous.

One important challenge for MBSE of CPS therefore is to enable the engineers to trace the different elements of a CPS along its development cycle back to the requirements. For the tools that are used for the development, this means that they require a common approach to provide and process the traceability-relevant data, and to have a common syntax and semantics for generating and transmitting the data.

During the past decade, the Open-Services for Lifecycle Collaboration (OSLC) specifications (Open-services.net, 2008) have emerged for integrating development lifecycle tools (e.g., modeling tools, change management tools, requirements management tools, quality management tools, configuration management tools) using Linked Data (Heath and Bizer, 2011) approach. The goal of OSLC is to make it easier for tools to work together by specifying a minimum amount of protocol without standardizing the behavior of a specific tool. The OSLC specifications use the Linked Data method to enable integration at the data level via links between artefacts. The artefacts information is represented as Resource Description Framework (RDF) (Manonla and Miller, 2004) resources identified by HTTP URIs. OSLC also provides a common protocol for manipulating those RDF resources through standard RESTful (Richardson and Ruby, 2007) web services such as creation (HTTP POST) and retrieval (HTTP

GET), update (HTTP PUT) and delete (HTTP DELETE) operations on RDF resources.

The main contribution of this paper includes the followings. First, we demonstrate a Linked Data-based approach for traceability in the model-based design process for CPSs, and the implementation of this traceability approach in the respective tools. This enables recording and establishing the traceability links of model elements (e.g., requirements, activities, artefacts, modeling tools,) through a standardized interface and format using OSLC. Second, we build a flexible trace links ontology of artefacts between requirements, simulation models, FMUs, simulation results, and test results. Third, we validate the ontology through an example workflow with heterogeneous artefacts in systems engineering.

2 Background

While there have been significant efforts dedicated to the introduction of traceability, there are several challenges that prevent traceability from being used in all the cases where such an application would be beneficial.

First, for traceability to be accepted in industrial use, the overhead that is created by it, must be minimal. Therefore, most functions need to be automated, and well integrated into different tools. Due to the heterogeneity of software tools that are used in systems engineering, all tools need to be equipped with interfaces that exchange data in a unified format, so that the syntax and semantics of the traceability data are compatible. Different "traceability information models" (TIM) have been developed that define the data model used to describe the traceability links between different entities (Mustafa and Labiche, 2017). However, up to now, no universal TIM was established.

For the usability of traceability, considering the representation of the essential traceability data depending on the context is important (Li and Maalej, 2012). Here, visualisation of the data as matrices, lists or trees was explored.

This paper describes the concept and following implementation of traceability in the INTO-CPS project, that was running from 2015 until 2017 (Larsen et al., 2016a). In this project, an integrated tool-chain for model-based design of CPS was developed. On the tooling side, INTO-CPS covers abstract modelling in SysML with the Modelio[1] tool. Detailed modelling of the different parts of a CPS is done in the tools Overture[2], 20-sim[3] and OpenModelica[4]. System simulation is executed through a newly created Co-Simulation Orchestration Engine (COE) based on the FMI standard (Blochwitz et al., 2012). The FMI standard allows creation of executable units, called Functional Mock-up Units (FMUs) that contain a simulation model and its solver. Such an FMU can be controlled with a standardized API, and its input and output signals are described with an XML file, which is called the `modelDescription.xml`. Test automation and model checking are performed by the RT-Tester toolsuite[5]. INTO-CPS enables a smooth workflow between the different tasks of CPS development, and the corresponding tools. Regarding traceability, the different tools initially had no interface for exchanging traceability data.

License rights for distribution and usage of INTO-CPS application, along with the COE and the traceability daemon and the SysML profile for Modelio, which also contains the traceability features, reside with the INTO-CPS association [6] under open source. The traceability functions for Overture and OpenModelica belong to the respective organisations and are also available under an open source license.

3 Traceability in the INTO-CPS project

3.1 Scope

The primary scope for traceability in INTO-CPS is demonstration of the basic traceability tasks across the tool-chain. This includes mostly tracing of the requirements and connecting them with the models, the simulation results, the produced code and test results. Furthermore, analysing the impact of changes (e.g. in requirements) to the overall system can be supported through traceability. Traceability is meant to create as little overhead for the user as possible, so that most actions should occur automatically without the need for user interaction. The openness of the INTO-CPS tool-chain shall be maintained, by defining open interfaces and formats, such as the ones described in this paper.

However, it is not in the primary scope of the traceability work in INTO-CPS to be able to trace back all steps of the development and revert to each step in the development's history. For this, other parts of the INTO-CPS project and tools are seen as more appropriate, such as versioning tools (SVN, Git) or functionality of the INTO-CPS Application. Therefore, as will be described below in Section 5, some of the tools support Git for versioning.

3.2 Ontology

The traceability ontology for INTO-CPS uses concepts from the PROV working group of the World Wide Web Consortium (W3C)[7]. PROV deals with three objects: *entity*, *activity* and *agent*. Entities can include requirements, models, simulation configuration files or simulation results. Activities can be saving of a model, running a co-simulation or more. An agent is a user that performs these activities.

[1] see `https://www.modelio.org/`
[2] see `http://overturetool.org/`
[3] see `http://www.20sim.com/`
[4] see `https://openmodelica.org/`
[5] see `https://www.verified.de/products/`
[6] see `http://into-cps.org/`
[7] see `https://www.w3.org/TR/prov-overview/`

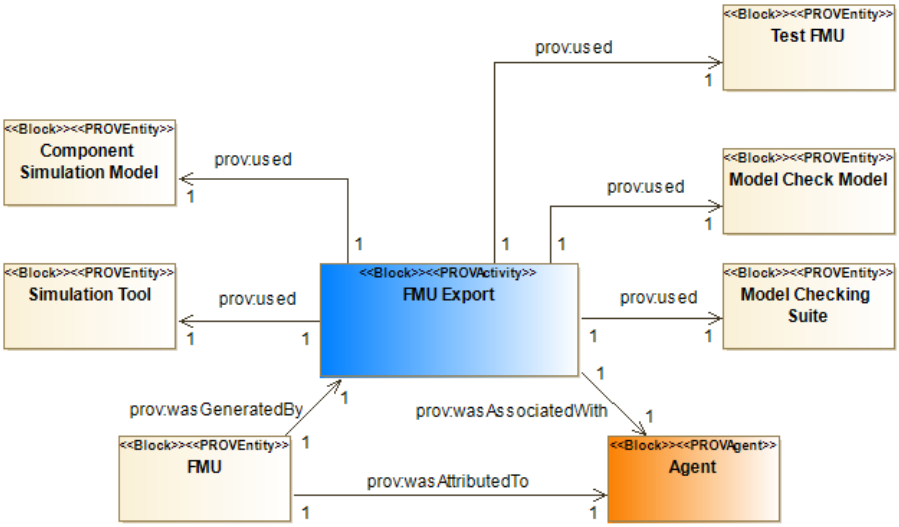

Figure 1. Block Definition Diagram showing the *FMU Export* activity.

In the context of traceability, these objects are connected by relations. PROV proposes relations such as "used", "was generated by" or "was derived from". In addition to these relations, further relations are used for the ontology of INTO-CPS, to express the relevant relations. These are taken from the OSLC[8] definitions. Furthermore, custom relations are proposed. The complete list of relations is described in the following table:

`prov:used`	one entity used another one
`prov:wasAttributedTo`	attribution of entities
`prov:wasAssociatedWith`	association of activities
`prov:wasGeneratedBy`	one entity is generated from another
`prov:wasDerivedFrom`	one entity is derived from another
`prov:hadMember`	one entity has one or more members
`oslc:elaborates`	an entity that elaborates on a requirement
`oslc:satisfies`	an entity that satisfies a requirement
`oslc:verifies`	an entity that verifies an assumption
`into:doesNotVerify`	an entity that does not verify an assumption
`into:violates`	an entity that violates an assumption

A number of activities are defined which shall create traces, using the objects and relations shortly described above. To illustrate this concept, a single activity is described in the following Figure 1 with its SysML Block Definition Diagram. The Activity "FMU Export" is associated with an agent and related to a number of entities. It generates an FMU (e.g. a file) and uses a simulation tool and a Component Simulation Model, and may in addition use a Test FMU, a Model Check Model and a Model Checking Suite, if the FMU is derived from a Model Checking activity. Finally, the FMU Export activity is associated with an Agent, and the FMU itself is attributed to this agent.

3.3 Architecture

This section introduces an outline of the tool and file system elements that support the traceability activities in the INTO-CPS tool-chain, and this outline is shown in Figure 2.

The central element of the traceability architecture is the *traceability daemon*, along with an interface (a REST-API[9]) that can be used in two ways: The tools (e.g. 20-Sim, OpenModelica, Overture, Modelio, RT-Tester, the COE, the INTO-CPS application) write data to the interface (e.g. they send traceability information from actions that happened within the tools to the daemon), the INTO-CPS application queries the interface (e.g. for retrieving information from the database). The daemon acts as front-end to the database (both for write and read operations, as explained later).

A schema [10] for the traceability messages is developed, which defines the format of the messages, and restricts their content. The schema defines the valid syntax of data submissions from the various 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

[8] see `http://open-services.net/`

[9] REST: Representational State Transfer; API: Application Programming Interface

[10] see `https://github.com/INTO-CPS-Association/into-cps-application/tree/development/src/resources/into-cps/tracability/schemas`

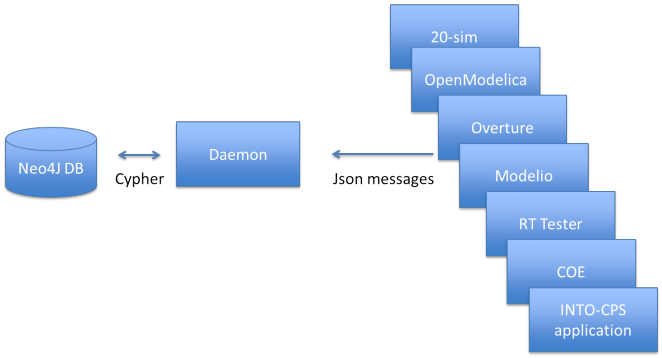

Figure 2. Schematic architecture of the traceability-related tools.

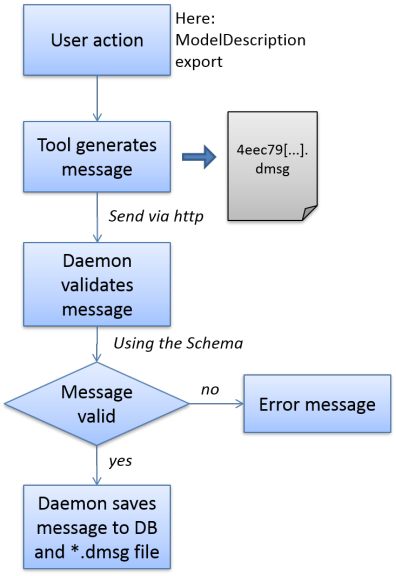

Figure 3. Schematic process of generating and saving a traceability message. For readability, the message content is omitted.

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 Section 3.2) is covered by it. The process of generating messages, sending them to the daemon and validating them, is shown in the Figure 3 below. The user action, which in this example is the export of a `modelDescription.xml` file from the Modelio tool, triggers the generation of a message with the file ending *.dmsg, which shall comply to the Schema. This message is sent via HTTP to the daemon, who validates the message according to the schema. If the message is valid, the daemon adds the content of this message to the global database and saves the *.dmsg file in the project folder.

4 Example Workflow

To illustrate the methods that have been described in the previous sections, we introduce an example. Figure 4 illustrates on the left side a complete workflow for the process from System Modelling, Behaviour Modelling and Co-Simulation, here performed using the tools Modelio, OpenModelica and the INTO-CPS Application..

The user starts with System Modelling, where the system is described by using SysML blocks including their ports and attributes. Requirements are written down and linked to the different SysML blocks that shall implement them. From a single block, a `modelDescription.xml` file can be exported, which is then imported into a modelling tool such as OpenModelica, to give the frame in which the behaviour will be implemented. This model will be saved, and finally exported to an FMU. Multiple of these FMUs will be connected in the INTO-CPS Application to form a Multi-model. The Co-Simulation will be set up with parameters such as time-stepping or simulation duration. Finally, the Co-Simulation is run to give the simulation traces.

The right hand side of Figure 4 shows the main artefacts, from the requirements over the model files to the simulation results, and their relations in the Prov or OSLC notation (see Section 3.2), as they are stored in the traceability database. Note that for readability, we show only the most relevant artefacts here, and omit those that are created by the tools to represent authors, activities, tools, timestamps or additional information. The figure shows that all the artefacts are linked, so that it is for instance possible to see that a particular requirement has been implemented in a system element, and later its behavior has been simulated. As we will discuss in Section 6, it is also possible with the usage of a test automation tool, to add information about the verification or violation of requirements.

This example only shows a very simple workflow, without advanced activites such as Model Checking or Design Space Exploration. It does, however, illustrate the relation between the engineering workflow, the traceability data and its representation. The different steps in the development cycle of a CPS that are depicted in Figure 4 are usually performed by multiple people or organisations, using different tools.

5 Implementation in the INTO-CPS tools

This section describes the implementation of the traceability features in the different tools, which are shown in Figure 2. OSLC is chosen as an implementation specification to manage traceability information from different lifecycle tools used in the model-based design of Cyber-Physical Systems. In our prototype, artefacts and their

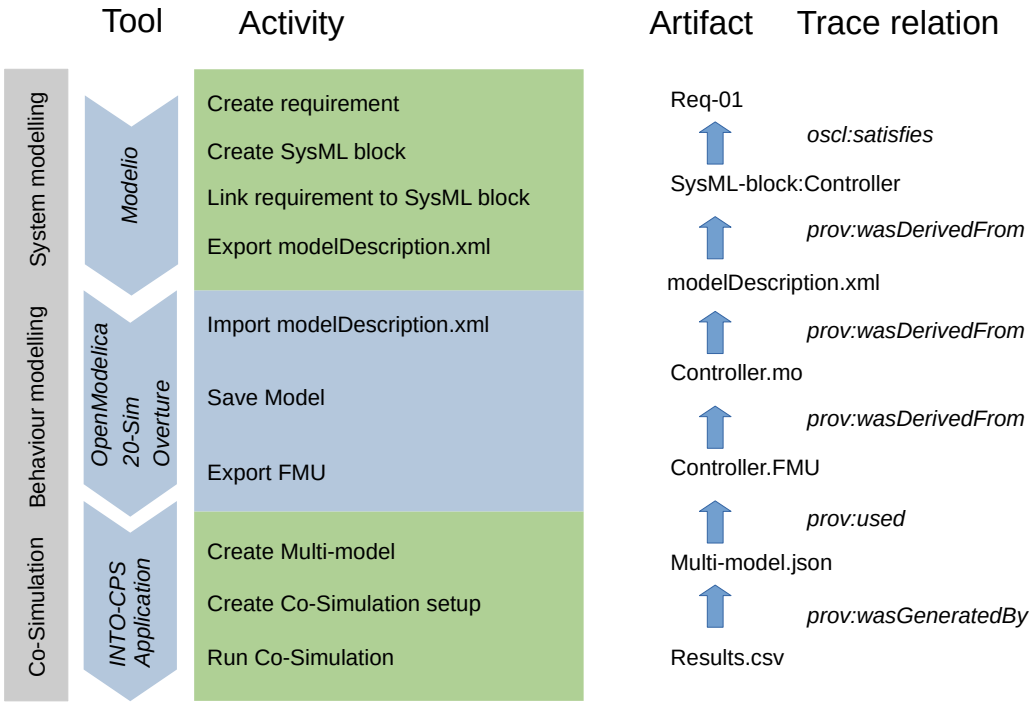

Figure 4. Example workflow (left), and most relevant artefacts and their trace relations (right).

relationships are described using an RDF/JSON format. The structure of the RDF consists of RDF triples with a Uniform Resource Identifier (URI) grouped into graphs: artefacts(subjects), relations (Predicates) and what it belongs to (objects). For instance, "Model (Subject) satisfies (Predicate) a requirement (Object)". The type of link (i.e., the predicate used in the link triple) defines the semantics of the link thus providing traceability between artefacts.

5.1 Traceability Daemon

The *traceability daemon* is essentially the core of the traceability tool support. In our implementation, it is launched or terminated by the INTO-CPS Application. The daemon's primary function is to create an OSLC compliant HTTP port and listen for the POST and GET actions. The different tools send data to an IP address at which the daemon is running (which, in our implementation, must be known to the tools). It stores the data sent via a POST request and will return a suitable response to a GET request. The requests are sent from the different tools, and the received data is stored in the Neo4J database. The daemon provides an interface that allows to retrieve the required data. This interface basically passes Cypher (see Section 6) queries to the Neo4J database.

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. To solve this, a step is added between receiving of the traceability messages and storing them in the Neo4J database. Each message the daemon receives is saved as plain text into a single file (with .dmsg file ending) in the project folder. The content of one such file is indicated in Figure 3. At the 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 global repository. Merging of the database is then done 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 5 below.

In addition, the daemon is validating the messages it receives from the tools with respect to the schema, 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 6) return meaningful information.

5.2 Modelio

The requirements definition part is handled only by Modelio. Modelio represents the *Architecture Modeling* activity in the INTO-CPS workflows. Modelio records the following traceability actions: Architecture creation, Architecture modification, ModelDescription export, Co-Simulation configuration export, Requirements generation and linking to SysML blocks.

Consequently, these actions are traced[11]. The Architecture creation / modification captures the generation and modification of a SysML block in Modelio. The genera-

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

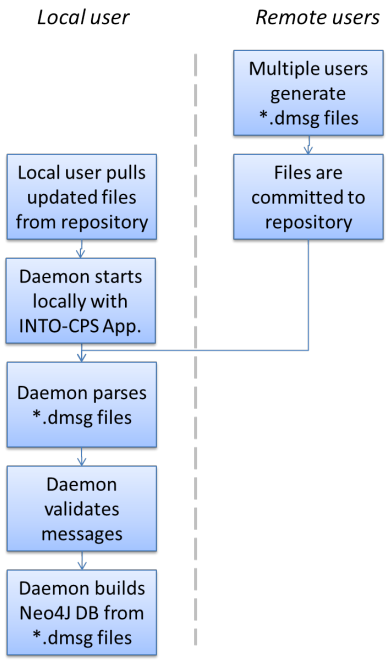

Figure 5. Schematic process of merging multipe messages from different users and building the Neo4J database from them.

tion of `modelDescription.xml` files from a SysML block is the next step in the workflow. Exporting a co-simulation configuration from a SysML connections diagram, which can be 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 *oslc:verifies* or *oslc:satisfies*.

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 situations, where traceability was not used from the very beginning.

5.3 Modeling tools

While Modelio is the starting point for requirement tracing, the modeling tools OpenModelica, 20-sim and Overture capture specific functional and non-functional aspects of the system design. They are described briefly in this section. Generation of models, either from scratch or from an imported `modelDescription.xml` file (e.g. coming from Modelio in the previous 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 modifiation, FMU export, FMU import, modelDescription.xml import.

20-sim 20-sim is a modeling and simulation tool for mechatronic and control systems. 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 also traced. 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.

OpenModelica OpenModelica is an open-source modeling and simulation tool which uses the Modelica language (Fritzson et al., 2018). Traceability support in OpenModelica is very similar to the one implemented in 20-sim. After an initial configuration of the Git repository and traceability daemon, the actions for saving a model, import of a `modelDescription.xml` file and export of an FMU are traced without further user interaction. Traceability in OpenModelica is described in more detail in (Mengist et al., 2017).

Overture In Overture, which is a modeling tool using the Vienna Development Method (Larsen et al., 2010), traceability is implemented as an additional package (as a `.jar` file), that can be downloaded from the GitHub page [12]. 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.

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

5.4 RT-Tester

RT-Tester is a tool for test automation and model checking. It 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.

5.5 INTO-CPS Application

The INTO-CPS Application is the graphical user interface for configuring and running co-simulation scenarios, design-space exploration, test automation and model checking (Larsen et al., 2016b). It 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 a Co-Simulation configuration from a multi-model, or modifies these configurations. Finally, the start of a simulation run is also recorded.

6 Queries 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 6.2. These categories can be extended and minimized, to present a neatly arranged view to the user. 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, using the Cypher query language (see Section 6.1 below).

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 (Winkler and Pilgrim, 2010), which displays traceability relations between requirements on different levels (e.g. high-level requirements and their refinements) as trees or lists [13]. Another common 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 standard way of presenting links is the graph view, where the different artefacts and their relations are shown in a graph. This is possible using the built-in Neo4J interface which allows browsing the complete graph. 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.

6.1 Cypher query language

The Neo4J database uses a query language called *Cypher*. This language uses ASCII syntax 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 [14]. In a study by Rath et at, graph query languages such as Cypher were also identified as suitable for requirements traceability (Rath et al., 2017).

6.2 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 search queries generate lists of items, which can be minimized to keep an overview of all the presented data.

The following queries are implemented in the INTO-CPS Application to allow for an 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 6). Clicking on "Trace Objects" shows the overview of the different queries, that can then be extended and minimized.

FMU and requirements This query is ran 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 5.3.)

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

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

[14] `https://neo4j.com/developer/cypher-query-language/`

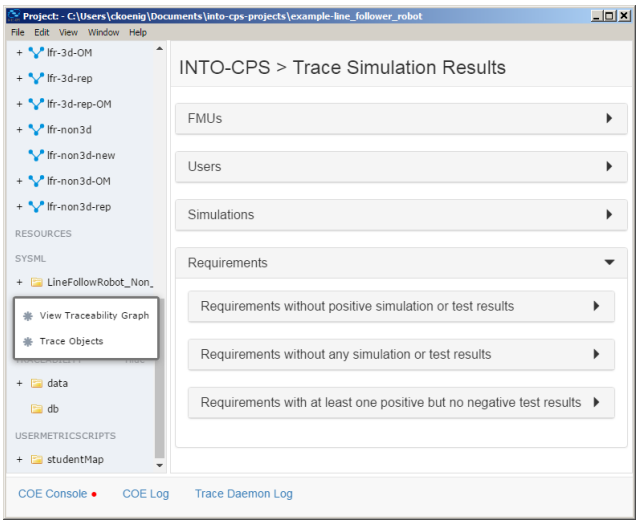

Figure 6. Overview of the traceability queries in the INTO-CPS Application.

```
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 the following command (note that "act" denotes an activity, and "elem" denotes an element):

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

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

Users, 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 using 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.user@mail.com*) can be found by:

```
match (usr{uri:'Agent.user@mail.com'})<-
  [: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 activities can be found in the schema under the enumeration for `ArtefactType`. In addition, all the activities performed by this user can be traced by:

```
match (usr{uri:'Agent.user@mail.com'})<-
  [:Trace{name:'prov:wasAssociatedWith'}]-
  (entity)
return entity.uri, entity.type
```

The activities are for example architectureModelling, modelDescriptionExport, simulationModelling and so forth. A complete list of activities can be found in the schema under the enumeration for `ActivityType`. Those activities reflect the activities as described in the ontology (see Section 3.2).

Simulation results and files To show which resources were used to generate simulation results, all the simulation results are queried first 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 a particular 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 relate requirements with the test results from RT-Tester (see Section 5.4), 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 the following command:

```
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, and therefore were neither found to violate a requirement, nor to verify one. 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.

6.3 Evaluation of the current tool-chain

While the presented implementation in the different tools is mainly a proof of concepts, some conclusions for a general evaluation of the concepts can be drawn. The open architecture that relies on OSLC allows different tools, independent of their platform, to exchange data through well-established standards (such as HTTP or JSON). The format of the messages is published in a JSON schema file, which allows any other tool to verify its messages and connect to the workflow. The overhead in terms of the amount of data that is being sent and stored is considered to be little, as the JSON files are lightweight, and only relevant activities are recorded. The Neo4J database with its Cypher querying language allows flexible querying of the database, so that additional activities or tools could easily be integrated. The current solution with the traceability daemon running locally with the INTO-CPS application should in future be replaced with a centralised traceability service, which is however easy to implement. Therefore, the tool prototypes currently implement no dedicated error handling if no connection to the daemon is available. Furthermore, no mechanism to handle user authentification was implemented in our prototype. In summary, we consider our approach to traceability to be simple and yet powerful enough, due to its openness and little overhead. Especially for larger project in safety critical domains, where documentation of the relation between requirements, tests and validation results is required, the presented approach is promising.

7 Related Work

There are several existing techniques that aim to model traceability among heterogeneous domains. However, in a systematic literature review conducted in (Mustafa and Labiche, 2017), existing traceability approaches are either limited to a specific domain and problem, or they lack to specify traceability link semantics. For example, a Traceability Information Model (TIM) has been proposed in (Taromirad et al., 2013) for capturing heterogeneous artefacts in the context of the safety-critical systems domain where the TIM is defined on top of multi-domains using Ecore (Steinberg et al., 2009) metamodeling language. This model, however, lacks to specify how source and target artefacts can be linked and it is not clear how to classify a trace link or an artefact.

In (Hung Le Dang and Hubert Dubois and Sébastien Gérard, 2008), the authors proposed a traceability model for tracing heterogeneous artefacts (requirements model, design artefacts, and verification and validation model) in automotive systems. This solution only supports specific types of trace links and cannot be extended to support a new traceability link and various modeling languages.

The Traceability Metamodeling Language (TML) is also presented in (N. et al., 2009) for defining the syntax and semantics of traceability metamodels. It can support any type of traceability links including the newly created ones. However, only artefacts from Meta Object Framework (MOF)-based models can be traced and linked.

ModelBus (Hein et al., 2009) is a tool integration framework in the domain of system engineering, which uses traceability data such as timestamps for artefact creation and modification, and information about the creator of a specific artefact. But, it rather uses a one to one transformation for the integration of two tools since there is no common data format. As with ModelBus, our approach also builds the integration based upon Web Services. However, the usage of common data format and OSLC in our approach makes the integration more up to date to the latest industrial standard of managing traceability data in the whole development lifecycle in the model-based design of CPSs.

Compared to the above existing approaches, we presented a linked data-based approach to handle standardized traceability links for heterogeneous artefacts from different lifecycle modeling languages and simulation tools. The integration is based upon the standardized defined schema to ensure that all tools use the same format for sending their data, and an ontology was defined to describe the data that is collected at different events.

8 Conclusion

This paper presents the results of the traceability and model management efforts in the INTO-CPS project. A common architecture for traceability was designed, using a central database as repository for all traceability information, and a daemon to receive data from the different tools. A message schema was defined that ensures that all tools use the same format for sending their data, and an ontology was defined to describe the data that is collected at different events. Collaborative work involving multiple users is supported. All tools record the relevant actions, and the whole workflow of INTO-CPS is covered, with respect to traceability data. Collection of data is automated as far as possible, minimizing overhead for the users. 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, 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 interface, ontology and format for the messages are public, 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.

In the context of INTO-CPS, we enabled requirements traceability in the whole tool-chain of CPS design, from requirements collection, systems modeling, through 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.

Acknowledgment

Support for the INTO-CPS project through the European Commission's Horizon2020 funding scheme (Grant Number 644047) is gratefully acknowledged.

References

T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *Proceedings of the 9th International Modelica Conference*, Munich, Germany, September 2012.

Peter Fritzson, Adrian Pop, Adeel Asghar, Bernhard Bachmann, Willi Braun, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Alejandro Danós, Rüdiger Franke, Mahder Gebremedhin, Bernt Lie, Alachew Mengist, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, Volker Waurich, and Per Östlund. The openmodelica integrated modeling, simulation and optimization environment. In *Proceedings of the 1st American Modelica Conference*. Modelica Association and Linköping University Electronic Press, October 2018. doi:10.3384/ecp18154206.

Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 2011. doi:10.2200/S00334ED1V01Y201102WBE001.

Christian Hein, Tom Ritter, and Michael Wagner. Model-driven tool integration with modelbus. In *Proceedings of the 1st International Workshop on Future Trends of Model-Driven Development*, pages 35–39. SciTePress, May 6-10 2009. doi:10.5220/0002174800350039.

Hung Le Dang and Hubert Dubois and Sébastien Gérard. Towards a traceability model in a marte-based methodology for real-time embedded systems. *Innovations Syst Softw Eng*, 4 (3):189–193, 2008. ISSN 1614-5054.

Peter Gorm Larsen, Nick Battle, Miguel Ferreira, John Fitzgerald, Kenneth Lausdahl, and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *SIGSOFT Softw. Eng. Notes*, 35(1):1–6, January 2010. ISSN 0163-5948. doi:10.1145/1668862.1668864. URL `http://doi.acm.org/10.1145/1668862.1668864`.

Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzson, Joerg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Sylianos Basagiannis, and Andrey Sadovykh. Integrated tool chain for model-based design of cyber-physical systems: The into-cps project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, Vienna, Austria, April 2016a. IEEE. http://ieeexplore.ieee.org/document/7496424/.

Peter Gorm Larsen, Casper Thule, Kenneth Lausdahl, Victor Bandur, Carl Gamble, Etienne Brosse, Andrey Sadovykh, Alessandra Bagnato, and Luis Diogo Couto. Integrated Tool Chain for Model-Based Design of Cyber-Physical Systems. In Peter Gorm Larsen, Nico Plat, and Nick Battle, editors, *The 14th Overture Workshop: Towards Analytical Tool Chains*, pages 63–78, Cyprus, November 2016b. Aarhus University, Department of Engineering. ECE-TR-28.

Yang Li and Walid Maalej. Which traceability visualization is suitable in this context? a comparative study. In Regnell B. and Damian D., editors, *Requirements Engineering: Foundation for Software Quality. REFSQ 2012*, volume 7195 of *Lecture Notes in Computer Science*, pages 194–210. Springer, Berlin, Heidelberg, 2012.

Frank Manonla and Eric Miller. RDF Primer. W3C Recommendation. World Wide Web Consortium. `https://www.w3.org/TR/2004/REC-rdf-primer-20040210`, 2004. [Online; accessed 10-January-2020].

Alachew Mengist, Adrian Pop, Adeel Asghar, and Peter Fritzson. Traceability support in openmodelica using open services for lifecycle collaboration (oslc). In *Proceedings of the 12th International Modelica Conference*. Modelica Association and Linköping University Electronic Press, May 2017. doi:10.3384/ecp17132823.

Nasser Mustafa and Yvan Labiche. The need for traceability in heterogeneous systems: A systematic literature review. In *41st International Computers, Software and Applications Conference*. IEEE, July 4-8 2017. doi:10.1109/COMPSAC.2017.237.

Drivalos N., Kolovos D.S., Paige R.F., and Fernandes K.J. Engineering a dsl for software traceability. In Gašević D.and Lämmel R. and Van Wyk E., editors, *Software Language Engineering*, volume 5452 of *in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*, pages 151–167. Springer, Berlin, Heidelberg, 2009.

Open-services.net. Open services for lifecycle collaboration – lifecycle integration inspired by the web. `http://open-services.net`, 2008. [Online; accessed 10-January-2020].

Michael Rath, David Akehurst, Christoph Borowski, and Patrick Mäder. Are graph query languages applicable for requirements traceability analysis? In *REFSQ Workshops*, 2017.

Leonard Richardson and Sam Ruby. *RESTful Web Services (First ed.)*. O'Reilly, 2007.

Houbing Song, Danda B. Rawat, Sabina Jeschke, and Christian Brecher. *Cyber-Physical Systems: Foundations, Principles and Applications*. Academic Press, Inc., Orlando, FL, USA, 2016. ISBN 0128038012, 9780128038017.

David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley, Boston, MA, 2009.

Masoumeh Taromirad, Nicholas Matragkas, and Richard F. Paige. Towards a multi-domain model-driven traceability approach. In *7th International Workshop on Multi-Paradigm Modeling co-located with 2013 ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 27–36. MPM@MoDELS, September 2013.

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.