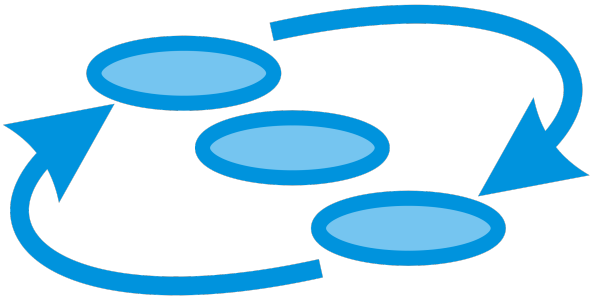
Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs



INTO-CPS

Methods Progress Report 1

Deliverable Number: D3.1b

Version: 1.0

Date: 2015

Public Document

Contributors:

John Fitzgerald, UNEW
Carl Gamble, UNEW
Richard Payne, UNEW
Ken Pierce, UNEW

Editors:

Richard Payne, UNEW

Reviewers:

Christian König, TWT
Andrey Sadovykh, ST
Claes Dühring Jaeger, AI

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	03-08-2015	Ken Pierce	Draft structure of deliverable and responsibilities
0.2	18-09-2015	Richard Payne	Draft concepts added
0.3	28-10-2015	Richard Payne	Concepts updated for comments; version for use by rest of project
0.4	13-11-2015	Carl Gamble	PROV and OSLC concepts, ontology and examples added
0.5	13-11-2015	Richard Payne	Traceability and provenance SotA added
0.6	15-11-2015	Carl Gamble	DSE and Traceability progress statements added
0.7	16-11-2015	John Fitzgerald	Introduction added
0.8	16-11-2015	Ken Pierce	Requirements and methods SotA added
1.0	15-12-2015	Carl Gamble	Internal review comments addressed

Abstract

This document reports progress in Work package 3 (Multi-modelling Methods) in the first year of INTO-CPS. It also reports on the project's survey of the state of the art in multi-modelling methods and heterogeneous system models, requirements engineering, and traceability and provenance for CPS design. An appendix is included that describes additional preliminary work on the use of a combination of W3C Prov and OSLC specifications for traceability and provenance within the INTO-CPS technologies.

Contents

1 Introduction

This report describes the work undertaken in Year 1 of INTO-CPS in Work Package 3. New and prospective users of the INTO-CPS technologies should refer to the Deliverable 3.1a Method Guidance 1 [FGPP15].

Work Package 3 (Multi-modelling Methods) in INTO-CPS aims to provide pragmatic methods in the form of guidelines and patterns that support the emerging tool chain. Our focus is on ensuring that adoption of the tool chain is cost-effective by providing guidance to help users determine the modelling technologies and patterns that best meet their needs and integrate with their work flows, taking into account their previous experience and processes. Ultimately, the full set of guidelines will cover the following areas:

- Workflows: multi-model construction from requirements capture in SysML, and the integration of multi-models into existing development activities and processes.
- Design Space Exploration: the use of co-simulation to support consideration of design options.
- Provenance and Traceability: methods for machine-assisted recording and maintenance of links between models, multi-models and other design artefacts.

The guidelines are underpinned by a common concept base, and are supported by a growing set of pilot studies that can serve as benchmarks for the methods and tools and as illustrations for the tool chain's capabilities.

The first release of the new multi-modelling tool chain was scheduled for late in Year 1. Our priority in WP3 was therefore to lay foundations for guidelines that will emerge from experience with the tool chain, and to provide specifications for functionality required of tools in the areas of Design Space Exploration (DSE) and traceability. Our specific objectives were therefore as follows:

- To survey the state of the art in multi-modelling methods, so that the project could identify promising techniques and tools and position its own contributions. This survey is reported in Section 3.
- To review the workflows in the case study partners and identify a candidate initial workflow against which the INTO-CPS tool chain could be validated once released. This was undertaken in Task 3.1 and is reported in Section 2.1.
- To specify the Design Space Exploration (DSE) support that would ultimately be required in the toolchain. This was undertaken in Task 3.2 and is reported in Section 2.2. The specification will be implemented in Task 5.1, for which Year 1 progress is reported in Deliverable 5.1a [GHJ+15].
- To specify the traceability and provenance functionality that the toolchain should support. This would be implemented in Task 4.4 from Year 2 onwards. This was undertaken in Task 3.3 and is reported in Section 2.3. Details of the support for provenance features are given as Appendix A. The specification will be implemented in Task 4.4 from Year 2 onwards.
- To prepare a common concept base for the project, and deliver a first set of guidelines for the construction of SysML models for multi-modelling given different entry

points, domain knowledge and previous multi-modelling experience. This was undertaken in Task 3.4 and is reported in Section 2.4. The concept base is reported in D3.1a Method Guidance 1 [FGPP15].

- To document initial pilot studies that illustrate various properties of CPSs and features of the INTO-CPS technologies. This was undertaken in Task 3.5 and the initial pilot studies are described in reported in Section 2.5.

2 Progress on WP3 Tasks

2.1 T3.1: Workflows

In the first year of the project, the focus of this task was to propose a candidate initial workflow for the INTO-CPS technologies. Since the initial tool release was not planned until the end of this first year, full validation of the proposed workflow will not be achieved until later in the project. The main validation effort in this first year therefore entailed aligning the proposed workflows with both expected tool functionality, and in particular with the existing practices, needs and expectations of the industrial case study owners. In order to achieve this, two surveys were carried out through questionnaires, with the results informing the workflows work. These surveys were:

- the capabilities of the baseline tools
- the existing practices of the case study owners

The key result gathered from these surveys is that there will be no single, one-size-fits-all workflow for INTO-CPS. The scope of the tool chain is wide, and the experience and practices of the industrial case study owners are diverse. For this reason, we developed the idea a set of *activities* carried out during CPS development, which then form *workflows* when grouped and linked.

We have identified an initial set of activities from the surveys, which are presented as part of the method guidelines in Deliverable D3.1a [FGPP15]. These activities are also linked with the traceability artifacts described in Appendix A. From these activities we consider how two existing workflows —based on responses to the survey of industrial case study partners— could be enhanced by the INTO-CPS technologies. The work here also informed the "Example Use Case for the INTO-CPS Technologies" which presents a putative workflow that covers all aspects of the tool, though which is unlikely to be followed to the letter in practice.

As experience is gained with the tool chain, and feedback received from industrial case study owners, further workflows will be developed and guidelines will be created that advise on when and how each workflow could be followed. This follows on from existing work from the DESTECS baseline project [FLPV13a]. In the remainder of this section we present the key findings from the survey of industrial case study owners.

2.1.1 Summary of Results from the Survey of the Industrial Case Study Owners

The survey of the industrial case study owners (CLE, AI, UTRC, and TWT) was undertaken using a questionnaire filled in by each of the partners. The questionnaire was divided into six sections:

- Workflows and Requirements
- Modelling and Simulation
- Tracebility, Provenance, Model Management
- Design Space Exploration
- Collaboration and Distribution
- Testing, Prototypes and Realisations

From these survey, along with informal discussions, we are able to establish a 'profile' for each of the case study owners:

CLE Have well-established DE modelling practices based on formal proof and model checking. They have a small number of repeat customers. They are looking to INTO-CPS as a way to enhance their existing practice using simulation-based evidence.

AI Do not have well-established modelling- and simulation-based design processes. They are moving to developing both hardware and software in-house and are looking to adopt INTO-CPS to handle the complexity of CPS design.

UTRC Have well-established modelling practices in a variety of formalisms with some form of co-simulation. They have large libraries of reusable models. They are interested in using INTO-CPS to help establish better collaboration between existing design teams and to establish traceability through large CPS developments.

TWT Have well-established modelling practices focusing on software design using FMI co-simulation. They are interested in the hardware-in-the-loop testing based on INTO-CPS to improve confidence in designs.

Below we now summarise the key findings, commonalities and differences between the partners that inform these profiles and the workflows presented in Deliverable D3.1a [FGPP15].

Workflows and Requirements UTRC and CLE have standardised workflows and adhere to standards in their domains (building automation and railways respectively). In particular UTRC has a clear workflow of tasks undertaken in order by different engineers and they wish to paralellise this. AI are working towards meeting functional safety standard. UTRC and TWT follow high iterative approach to design. CLE follow a more linear V-model.

Microsoft tools are widely used to capture and manage requirements, in particular Excel (UTRC, CLE, TWT) and Word (AI). Additionally UTRC use the proprietary IBM

Rational DOORS[1].

UTRC and TWT identify difficulties in communicating terminology and vision between stakeholders when defining requirements for CPS designs. CLE and AI currently work with stakeholders within their own domains and do not experience as many problems currently.

All partners identified that design changes are triggered by *changes to stakeholder needs*, *problems encountered during design*, and *changes in budget*. These can lead to overruns, increased project cost, and increased product costs. Responses include tracking issues and evolutions through tickets, and conducting impact and feasibility analysis and evolving designs where appropriate.

Modelling and Simulation Models are manually created by all partners, though UTRC and CLE have some level of automatic model generation in specific circumstances. UTRC have a libraries of models for standard components and reuse is common practice.

TWT mainly use CT models. AI have some CT models of mechanical components (FEM) and use electrical models for documentation. AI do not currently model software using DE or state machines. CLE current modelling focus are software verification through DE models and state machines. UTRC use a wide variety of models, with current focus on Matlab, Modelica and state machines. Both TWT and UTRC do multi-domain modelling in Matlab/Simulink and co-simulate using FMI.

All parters identified areas where improved co-simulation would improve their workflows. All partners expressed a wish to have GUI interfaces for general use, with access to command-line scripting for specific use cases.

Tracebility, Provenance, Model Management The partners currently manage various levels of traceability through diverse tools. CLE and UTRC use Excel, while TWT use Fogbugs Project Tracker. Practices at AI are evolving from third party to in-house software development, but some traceability can be established from documentation. All partners identified a need to store and manage diverse traceability artefacts (including models, results, design notes and documentation).

UTRC use Mathworks' Validation and Verification suite in some cases to trace requirements between DOORS and models, but in general the diversity of engineering domains and lack of tool support leads to long face-to-face meetings to establish agreement on traceability. In addition high-level requirements can be realised through multiple components (or models thereof) leading to complex traceability needs.

The partners use various practices to help record authorship of artefacts including comments within models and records in version management software. All partners identify a need to produce certification evidence or customer documentation from traceability and/or provenance data recorded during a development.

[1] `http://www-03.ibm.com/software/products/en/ratidoor`

Design Space Exploration Current practice in DSE is typically carried out manually and relies on the tacit knowledge and expertise of engineers. UTRC however have some internals tools for carrying out DSE. No partner currently use exploration techniques that reduce design space size (e.g. Taguchi tables, simulated annealing).

TWT currently explore both CT model variants and parameters within those variants. AI expressed a clear desire to do this as well, in addition to sweeping DE parameters such as control thread periods.

Results are gathered in CSV/spreadsheets and are further processed in Excel and Matlab, and in Origin (UTRC) and Python (TWT). Visualisations include histograms, graphs and custom GUIs and are typically manually assessed through inspection, though TWT generate reports automatically.

Collaboration and Distribution All partners identified a need to pass artefacts between design teams (models and model fragments, simulation results, textual information, requirements information, and test cases). Partners with existing modelling practices (UTRC, CLE, TWT) perform integration testing.

Use of distributed computing is not a core part of existing practice. TWT perform some remote simulations. Use of distributed computing is of interest to UTRC but current solutions are too complex.

Testing, Prototypes and Realisations Partners with existing modelling practices (UTRC, CLE, TWT) perform automated testing of models and use of models for generating tests (and requirements in the case of TWT). These partners also use simulation results to inform models in an iterative way. UTRC in particular have a large body of existing testing practices. Test cases are produced manually by AI and CLE. In addition to testing, CLE and UTRC use other formal verification methods including model checking and formal proof.

Most partners (UTRC, CLE, AI) currently perform hardware- or software-in-the-loop simulation, and TWT are interested in adopting this practice. Additionally code generation is currently used by UTRC, TWT and CLE, typically targeting C.

2.2 T3.2: Design Space Exploration

In the first year of the DSE task the focus has been on assessing what the industrial partners currently understand of DSE, what their aspirations are for DSE within their case studies and what DSE approaches and algorithms should the INTO-CPS project aim to support. As outlined earlier in Section 2.1.1 the DSE approaches currently undertaken by the WP1 partners are manual and rely heavily on engineer expertise to both define product parameters and to analyse results. While there will always be a need for engineer expertise within the DSE methods to both define what aspects of a product design should be varied and how to measure simulation outputs to evaluate and rank designs, the tool support within INTO-CPS aims to support the engineer in several ways:

- reduce the workload in defining the parameters for and running multiple simulations to explore the design space by automating the configuration and launch of simulations;
- provide a range of DSE algorithms for the engineer to select from along with guidance about the suitability for different problem domains;
- provide templates and scripts to obtain objective measures of simulated CPS performance from the raw simulation results;
- support the use of both scripted ranking functions and pareto optimality to rank the results of each design globally;
- reduce the total number of simulations required to have confidence in finding a globally optimum solution by using design ranking and closed loop optimisation methods.

The details of both the currently implemented DSE method and the set of proposed methods may be found in D5.1a [GHJ+15]. This deliverable also contains the aspirations of the WP1 partners for DSE along with comment on how the proposed methods will meet them.

2.3 T3.3: Provenance and Traceability

The first year of provenance and traceability study has concentrated on exploring the both the relations that will be important to the record within the INTO-CPS tool chain and also potential standard notations and specifications that may be used to represent them. Two distinct specifications have been identified to form the foundations for the the provenance and traceability concepts within INTO-CPS, these specifications have the benefits of both being open and free to use, and, in case of OSLC, have a wide base of industry support. The W3C PROV [2] model provides support for recording the temporal relations between activities, entities and agents within a process. This supports the recording of, for example, links between simulation results and the models, platforms and configurations that produced them, this is important when generating documentation as part of a certification effort. To compliment this the relations specified by the OSLC [3] provides support for recording logical relations between objects within a data set. So OSLC allows the linking of, for example, a submodel and a requirement that it is designed to satisfy, or from a simulation result to the requirement it provides evidence for.

The key concepts of both PROV and OSLC are presented in Appendix A of this document along with the proposed provenance and traceability ontology, which will form the basis for data recorded by the various tools in the INTO-CPS tool chain. This appendix also contains example applications of both PROV and OSLC being used to represent many of the document relations expected when using the INTO-CPS tool chain and workflows.

[2] http://www.w3.org/TR/prov-overview/
[3] http://open-services.net

2.4 T3.4: Guidelines

In the first year of the project, the main aim of this task was to collate a concept base which will form a living document. The concept base [FGPP15] should be used as a resource for the project to ensure consistent terminology use when producing deliverables and also in dissemination materials (e.g. conference papers). The concept base used several inputs as baseline; primarily drawing on existing EU project outputs. These include:

TAMS4CPS TAMS4CPS[4] is a EU H2020 coordination and support action, concerned with an aim to identify research and development needs for modelling and simulation for CPSs. As such, the definitional framework produced in the project is a key input. Several general areas, such as definitions of CPSs, modelling and simulation were valuable inputs.

DESTECS The DESTECS[5] EU project was concerned with the co-modelling and co-simulation of embedded systems. As such, concepts including models, simulation and co-simulation were of use. However, as DESTECS focussed on embedded systems, several of the terms required lifting to the CPS domain. Several other technologies and tools used in INTO-CPS were not covered in DESTECS.

COMPASS The COMPASS[6] FP7 project focussed on the modelling and analysis of Systems of Systems (SoSs). SoSs exhibit several characteristics in common with CPSs, and as the COMPASS project used several technologies in common with INTO-CPS baseline (SysML, RT-Tester) the COMPASS concept and terminology base was of use.

The concept base, presented in Deliverable D3.1a [FGPP15], was circulated to the project partners for input to determine any areas of disagreement and for reference. Several terms were addressed, but at the time of writing, no major concerns have been raised. The concept base will therefore be used to inform a hyperlinked resource for the project in the next phases of the project.

The second aim of the task is to produce guidelines for multi-modelling. In the first year, we concentrated on guidelines for SysML modelling, presented in D3.1a [FGPP15]. These guidelines focus on the construction of SysML models for multi-modelling given different entry points, domain knowledge and previous multi-modelling experience. The guidelines make use of the INTO-CPS SysML profile in Deliverable D2.1a [APCB15].

2.5 T3.5: Pilot Case Studies

In Deliverable D3.4 [FGP+15] we present a collection of pilot studies that illustrate different aspects of the INTO-CPS baseline and future technologies. These studies were chosen against selection criteria in terms of properties of CPSs and their ability to demonstrate the features of the INTO-CPS technologies. We also ensure that the studies must be easily explained and material should be made available for teaching/training purposes.

[4] `http://www.tams4cps.eu`
[5] `http://www.destecs.org`
[6] `http://www.compass-research.eu`

In this section, we outline those CPS properties and INTO-CPS technologies, and describe how the chosen pilot studies meet the criteria. It is important to note that in this first year, we don't expect to have complete coverage of the criteria. Therefore in D3.4 we propose a roadmap for the next 12 months of case study and example development to test and demonstrate upcoming INTO-CPS technologies.

The INTO-CPS concept base in [FGPP15], describes CPSs as being "ICT systems (sensing, actuating, computing, communication, etc.) embedded in physical objects, interconnected (including through the Internet) and providing citizens and businesses with a wide range of innovative applications and services". As such, the pilot studies should exhibit *cyber*, *physical* and *network communication* characteristics. The pilot studies in this document address mainly address the first two of these CPS characteristics. The reason for this is that these examples test the baseline tools, which emphasise the modelling of cyber and physical systems and the co-modelling of these in embedded systems.

In the first year, the main criteria for technology coverage was to model the example with baseline tools. As such, we required the examples to be modelled with: Crescendo (Overture and 20-Sim), OpenModelica and SysML. In addition, it was deemed beneficial if the examples could be used to demonstrate technologies becoming available through the first year of the project. As such, Table 1 shows the baseline technologies used in the examples, and Table 2 show the INTO-CPS technologies available during year 1 that have been demonstrated by the pilot studies.

	Baseline			
Multi-model	Crescendo	OpenModelica	'Holistic' SysML model	RTTester
Three-tank Water Tank	x		x	
Fan Coil Unit (FCU)	x	x		
Line-following Robot	x	x	x	
Turn Indicator				x

Table 1: Overview of baseline technologies used for pilot studies

3 State of the Art in Multi-model Methods

In this section, we survey related projects in several areas of interest in WP3. These areas are: requirements engineering in Section 3.1; traceability and provenance in Section 3.2; design space exploration in Section 3.3 and in multi-modelling methods in Section 3.4.

Multi-model	INTO-CPS Technology													
	Multi-DE model	Multi-CT model	20-Sim (for FMU)	OpenModelica (for FMU)	VDM-RT (for FMU)	INTO-CPS SysML	Co-simulation Engine (COE)	SysML requirements	Traceability links included	Provenance graph included	DSE support included	Test Automation support	Model checking	SiL/HiL enabled
Three-tank Water Tank		x	x		x	x								
Fan Coil Unit (FCU)			x	x	x	x								
Line-following Robot		x	x	x	x	x								

Table 2: Overview of INTO-CPS technologies used for pilot studies

3.1 Requirements Engineering

The ADVANCE[7] project (along with the Deploy project, covered in the next section) is based around the Rodin tool and the Event-B formalism. ADVANCE extended Rodin to allow for FMI co-simulation between Event-B models and arbitrary FMUs. The Rodin tool supports ProR for managing textual requirements including hierarchies and links. The tool is now part of the Eclipse incubator program.

The COMPASS[8] project developed an approach for the development of Architectural Frameworks (COMPASS Architectural Framework Framework - CAFF). This was used to develop guidelines for SoS requirements modelling [9] called SoS-ACRE. Although the CAFF is agnostic with respect to modelling language, the guidelines were realised in SysML and insight from this fed into the SysML guidance given in Deliverable D3.1a [FGPP15].

The SPEEDS[10] project developed a Contract Specification Language (CSL), allowing collaborating teams to define requirements and promises about the components that their team is developing. This was extended in the DANSE[11] project to become the Goal and Contract Specification Language (GCSL), which permits architectural aspects to be captured in OCL[12] (Object Constraint Language) notation. DANSE methodology documents[13] advocate formalising requirements late in the architectural design workflow.

The TOPCASED project included TOPCASED-Req, a solution to manage requirement traceability in model for aviation, following the DO-178B lifecycle. The AGeSys[14] Project improved TOPCASED-Req to move away focus on the whole requirements lifecycle as

[7] http://www.advance-ict.eu/
[8] http://www.compass-research.eu/
[9] http://www.compass-research.eu/Project/Deliverables/D211.pdf
[10] www.speeds.eu.com
[11] http://danse-ip.eu/
[12] www.omg.org/spec/OCL
[13] http://danse-ip.eu/home/pdf/danse_d4.3_methodology_v2.pdf
[14] http://www.aerospace-valley.com/sites/default/files/encart_html/index.html#1

a new tool called becomes ReqCycle. TOPCASED and ReqCycle now form part of the PolarSys[15] open-source tools for embedded systems.

The META tool being developed under DARPA AVM (Adaptive Vehicle Make) programme makes use of the CyPhyML meta language. In terms of requirements they focus on "executable requirements" that are quantifiable tests that can be automatically checked against models or implementations. This has been demonstrated in OpenModelica.

The ENOSYS[16] project targeted design space exploration for FPGA design, using the MARTE UML profile for high-level specification. The Modelio tool was used in this project and therefore INTO-CPS already benefits from this research.

The MODRIO[17] and Openprod projects investigated mapping requirements to OpenModelica. Again as OpenModelica is a baseline tool for INTO-CPS, we can leverage these results as our technologies develop.

3.2 Traceability and Provenance

The *Deploy*[18] project presents an approach to traceability from informal requirements to a formal model in the Event-B notation [Pro12]. In contrast to INTO-CPS, Deploy concentrate on traceability from complete requirements to only state-based models, rather than the flexible tracing of system engineering model elements. As with the INTO-CPS project, Deploy see tool support as critical for traceability, and produce an integration of the ProR platform for requirements engineering and the Rodin platform for Event-B modelling. The *ADVANCE*[19] project continued this development. In addition, in collaboration with the *VERDE* project, a traceability solution was developed between Eclipse-based tools such as ProR and Topcased[20] for SysML. This uses a concept of *Tracepoints* to link to Eclipse model elements [GJ11].

In the *OPENCOSS*[21] project traceability is used for evidence management and impact analysis of requirement through to safety cases [vdBLK+15]. Several traceability relationships are defined linking 'artefacts'. The traceability links are then used as part of impact analysis using the Evidence Management tool support produced in the project. Using SVN repositories, a user must manually add artefacts, linked to resources in a SVN repository and record the necessary traces. This is in contrast to INTO-CPS which aims to have a central traceability and management tool which will be integrated with the INTO-CPS tool chain, and a greater number of traceability links covering a modelling and simulation.

In the *COMPASS*[22] project, requirements were modelled and there was support for traceability between models. Explicit traceability mechanisms supporting forwards and backwards tracing were not a focus of the project, however a traceability pattern was produced

[15] https://www.polarsys.org/
[16] http://www.enosys-project.eu/
[17] https://www.modelica.org/external-projects/modrio
[18] http://www.deploy-project.eu
[19] www.advance-ict.eu/
[20] https://www.polarsys.org/topcased
[21] http://www.opencoss-project.eu
[22] http://compass-research.eu

for SysML SoS engineering [PHP+14]. It would appear that provenance metadata could be captured within the existing framework, but this was not a subject of study in the project.

In common with INTO-CPS, the *SPRINT*[23] project uses OSLC to link models form various tools, from requirements in DOORS through to architectural models using IBM Rhapsody and Simulink and Modelica system models. Although not explicitly used for traceability or model management, INTO-CPS should make use of project results in the implementation and use of OSLC.

The *Openprod*[24] project lists an outcome as including "precise requirements capture and traceability based on behavior trees integrated with Modelica/UML in Eclipse; ontology-based generic 2D/3D graphic modeling and database coupling". However, we were unable to locate public deliverables.

In analysing the state of the art, the *CPSoS*[25] EU support action determined that in the automotive sectors, the collecting and managing maintenance and diagnostic data, dealing with heterogeneous data considering *provenance* and quality of data is a challenge to be addressed in the next 5 years [PEF+14]. The project also recognises the need for traceability from requirements through to implementation, but in their state of the art , they do not appear to identify any projects which tackle this issue [TPR+15].

In the *MODRIO*[26] project, traceability links were proposed between requirements, designs and scenarios for the purposes of requirement evaluation (understanding conflicted and redundant requirements) and impact analysis. A prototype traceability view was produced for ModelicaML, but in not extensible across a tool chain [Sch13].

The *Acosar*[27] project, starting in 2015, aims to integrate real-time systems into simulation environments. The project description concerns requirements, simulation and integration. It is not clear if traceability is a focus, however, results should be monitored.

TAPPS[28] are developing an open end-to-end tool chain for developing and deploying CPS Apps. It in unclear whether this will include traceability, and therefore project outputs should be monitored.

3.3 Design Space Exploration

A comprehensive overview of the state of the art in DSE is provided in Deliverable 5.1a [GHJ+15].

[23] `http://www.sprint-iot.eu`
[24] `http://www.ida.liu.se/labs/pelab/OpenProd/`
[25] http://www.cpsos.eu/
[26] `https://www.modelica.org/external-projects/modrio`
[27] `http://www.acosar.eu`
[28] `http://www.acosar.eu`

3.4 Multi-modelling Methods and Hetereogeneous System Models

The CRYSTAL[29] project (Artemis) is developing a framework for that allows OEMs to integrate tools for embedded systems development, based around the Crystal Interoperability Specification (IOS) V2.0, which is an extension of OSLC. The project extends work from previous ARTEMIS projects CESAR, iFEST and MBAT. Their method guidance is based on "Generic Engineering Methods" which are similar to the activities and workflows described in Deliverable D3.1a [FGPP15]. INTO-CPS should work to align our workflows with those identified by CRYSTAL, keeping in mind that the needs of the industrial case study owners is the first priority.

The TERESA[30] project aimed to provide guidelines for software process engineers to define trusted computing engineering processes in various domains (automotive, home control, industrial control, and metering) and in resource-constrained environments. INTO-CPS should investigate how these guidelines are presented and if they can inform any of the guidelines work for the INTO-CPS technologies.

Ptolemy II[31] offers a heterogeneous simulation framework for modeling DE and CT components within one model. The iCyPhy[32] is a pre-competitive industry-academic partnership pursuing well-founded methods for engineering of cyber-physical systems, including improvements to Ptolemy II. INTO-CPS should consider the gaps in current and research needs identified in Fisher et al. [FJLM14].

[29] http://www.crystal-artemis.eu/
[30] http://www.teresa-project.org/
[31] http://ptolemy.eecs.berkeley.edu/
[32] http://www.icyphy.org/

References

[APCB15] Nuno Amalio, Richard Payne, Ana Cavalcanti, and Etienne Brosse. Foundations of the SysML profile for CPS modelling. Technical report, INTO-CPS Deliverable, D2.1a, December 2015.

[FGP+15] John Fitzgerald, Carl Gamble, Richard Payne, Ken Pierce, and Jörg Brauer. Examples Compendium 1. Technical report, INTO-CPS Deliverable, D3.4, December 2015.

[FGPP15] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Method Guidelines 1. Technical report, INTO-CPS Deliverable, D3.1a, December 2015.

[FJLM14] Amit Fizher, Clas A. Jacobson, Edward A. Lee, and Richard M. Murray. Industrial Cyber-Physical Systems – iCyPhy. In M. Aiguier et al., editor, *Complex Systems Design and Management*, pages 21–37. Springer, January 2014.

[FLPV13a] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, and Marcel Verhoef. A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems. *To appear in Mathematical Structures in Computer Science*, 2013. Outdated by [FLPV13b].

[FLPV13b] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, and Marcel Verhoef. A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems. *Mathematical Structures in Computer Science*, 23(4):726–750, 2013.

[GHJ+15] Carl Gamble, Francois Hantry, Claes Dühring Jæger, Christian König, Alie El din Madie, and Richard Payne. Design Space Exploration in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D5.1a, December 2015.

[GJ11] Andreas Graf and Michael Jastram. Requirements, traceability and dsls in eclipse with the requirements interchange format (rif/reqif). Technical report, 2011. http://deploy-eprints.ecs.soton.ac.uk/307/1/mbees2011.pdf.

[LPH+15] Peter Gorm Larsen, Ken Pierce, Francois Hantry, Joey W. Coleman, Sune Wolff, Kenneth Lausdahl, Marcel Groothuis, Adrian Pop, Miran Hasanagić, Jörg Brauer, Etienne Brosse, Carl Gamble, Simon Foster, and Jim Woodcock. Requirements Report year 1. Technical report, INTO-CPS Deliverable, D7.3, December 2015.

[PEF+14] Radoslav Paulen, Sebastian Engell, Wan Fokkink, Haydn Thompson, Dagmar Marron, and Svetlana Klessova. Report about the first meeting of the Working Groups. Technical Report D1.2, EC FP7 project 611115 CPSoS, February 2014.

[PHP+14] Simon Perry, Jon Holt, Richard Payne, Jeremy Bryans, Claire Ingram, Alvaro Miyazawa, Luís Diogo Couto, Stefan Hallerstede, Anders Kaels Malmos, Juliano Iyoda, Marcio Cornelio, and Jan Peleska. Final Report on

SoS Architectural Models. Technical report, COMPASS Deliverable, D22.6, September 2014. Available at http://www.compass-research.eu/.

[Pro12] Deploy Project. Deploy methods: Final report. Technical Report D6.6, EC project 214158 Deploy, April 2012.

[Sch13] Wladimir Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool.* PhD thesis, Linkoping University, Department of Computer and Information Science. Linkoping University, The Institute of Technology., 2013. http://liu.diva-portal.org/smash/get/diva2:654890/FULLTEXT01.pdf.

[TPR+15] Haydn Thompson, Radoslav Paulen, Michel Reniers, Christian Sonntag, and Sebastian Engell. Analysis of the State-of-the-Art and Future Challenges in Cyber-physical Systems of Systems. Technical Report D2.4, EC FP7 project 611115 CPSoS, February 2015.

[vdBLK+15] Mark van den Brand, Luna Yaping Luo, Martijn Klabbers, Giorgio Tagliaferri, Vincenzo Manni Andrea Critelli, Jose Luis de la Vara, Sunil Nair, Carlo Ieva, and Rodolphe Arthaud. Evidence management service infrastructure: Methodological guide. Technical Report D6.7, OPENCOSS Project, 2015.

A Provenance and Traceability

A.1 Introduction

Traceability and Provenance to support requirements engineering, certification efforts and model management will be key features for INTO-CPS. In this appendix we present the initial work on using the W3C Prov concepts to support model management. The basic concepts of Prov are introduced first, before candidate specifications from the OSLC are presented. The PROV concepts and a selection of the OSLC relations are then combined into a proposed INTO-CPS ontology. Examples of how these concepts may be used to represent a range of anticipated activities within an INTO-CPS work flow are presented. We then discuss some findings found when trying to apply Prov to the UTRC WP1 case study before touching upon some open questions.

A.2 Prov Concepts

Provenance is the information about who was involved in producing something, the activities that took place and entities that were involved. The W3C PROV working group have defined a family of documents that describes the concepts in this domain and two in particular have been used in the development of the provenance and traceability work in INTO-CPS. The provenance ontology (PROV-O)[33] defines the concepts that are represented by PROV and this heavily influenced the construction of our traceability ontology presented later (Section| A.4). Alongside this the provenance notation (PROV-N)[34] has been used to produce concrete examples of the provenance data that would need to be recorded and from this we obtain the majority of the graphical views in this appendix.

In the following subsections we will present the main concepts of PROV-O, focussing on their relations and present examples in both the PROV-N notation and graphical form.

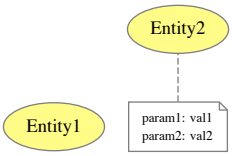
A.2.1 Entities, Activities and Agents

There are three node types in a Prov graph these are *entity, activity* and *agent.*

entities "An entity is a physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary". In the case of INTO-CPS, this would include requirements documents, models, simulation configuration files, simulation results and the simulators themselves.

activities "An activity is something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities". In INTO-CPS activities will include, modelling, simulation and co-simulation;

[33] `http://www.w3.org/TR/prov-o/`
[34] `http://www.w3.org/TR/prov-n/`

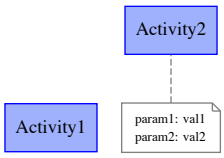


(a)

```
entity(Entity1)
entity(Entity2,[param1="val1", param2="val2"])
```

(b)

Figure 1: Prov entities in their graphic (a) and Prov-N (b) forms.



(a)

```
activity(Activity1)
activity(Activity2, 2015-05-18T16:00:00, 2011-05-18T17:00:00,
        [param1="val1", param2="val2"])
```

(b)

Figure 2: Prov activities in their graphic (a) and Prov-N (b) forms.

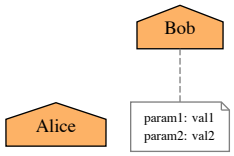
agents "An agent is something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity". Agent's will include individual engineers but could also include internal and external organisations such as the quality department or a component supplier.

The standard representation of Prov data is in the Prov-N notation and for each type there is a minimum set of data required along with one or more optional extras. Figure 1b shows the Prov-N used to create the two entities displayed in Figure 1a. Entity1 in both figures has the minimum allowed description, which is essentially just its unique id. Entity2 makes use of the optional list of key/value pairs by which Prov-N allows additional description data to be added. Similarly Figure 2b shows the Prov-N description of two activities, the first activity is simply named, while the second has both the optional start and end times defined along with two key/value pair parameters. The graphic representation of these two activities can be seen in Figure 2a where we may note that while the second activity has start and end times, these are not displayed. Finally Figure 3b shows the Prov-N description of two agents, one with just a name and the other with a pair of key/value parameters, their graphic representation can been found in Figure 3a.

A.2.2 Node Relations

Prov provides a set of relations that are used to describe the relationships between the three node types. The full set of relations may be found on the W3C PROV ontology [35],

[35] http://www.w3.org/TR/prov-o/



(a)

```
agent(Alice)
agent(Bob,[param1="val1", param2="val2"])
```

(b)

Figure 3: Prov agents in their graphic (a) and Prov-N (b) forms.

but in the example traces we have produced thus far we have only needed a subset of these. Starting then by considering only the entity and activity nodes, the key relations are:

used "Usage is the beginning of utilizing an entity by an activity. Before usage, the activity had not begun to utilize this entity and could not have been affected by the entity." As the quote implies, activities use entities, so we can imagine that simulation uses a model file.

was generated by "Generation is the completion of production of a new entity by an activity. This entity did not exist before generation and becomes available for usage after this generation." Entities are generated by activities, so we could say that a simulation result file was generated by the simulation activity.

was derived from "A derivation is a transformation of an entity into another, an update of an entity resulting in a new one, or the construction of a new entity based on a pre-existing entity." An 'entity ← was derived from ← entity' relation can reduce ambiguity found in an 'entity ← used ← activity ← was generated by ← entity' chain when there are multiple entities on both ends of the chain. For example if a modelling activity uses two input models and produces two output models, it is not clear which input models influenced which output models.

was informed by "Communication is the exchange of an entity by two activities, one activity using the entity generated by the other." This relation can be used to indicate communication between activities where the data communicated is either insignificant or transient. For example, this could be used to describe and OSLC exchange between two tools or the FMI communications between simulators and the COE.

The graphical and prov-n representations of these relations can be seen in Figure 4. It is worth noting at this point that the arrows point backwards in time and from the subject to the object. For example when recording the relation file X was generated by activity Y, the arrow will point to the activity.

Prov also includes relations to record responsibility for actions and entities, there are only three of these and all have proved to be of use when considering provenance examples for INTO-CPS:

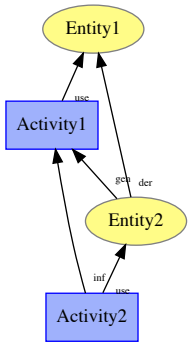
Attribution "Attribution is the ascribing of an entity to an agent." Very simply, at-

tributing an entity to an agent states who is responsible for it, this could mean the original author or the agent responsible for executing a simulation that produced it.

Association "An activity association is an assignment of responsibility to an agent for an activity, indicating that the agent had a role in the activity. It further allows for a plan to be specified, which is the plan intended by the agent to achieve some goals in the context of this activity." The association here could indicate the agent that was responsible for executing a co-simulation or for performing some modelling work. Thus far we have not required to use the plan element of association.

Delegation "is the assignment of authority and responsibility to an agent (by itself or by another agent) to carry out a specific activity as a delegate or representative, while the agent it acts on behalf of retains some responsibility for the outcome of the delegated work. For example, a student acted on behalf of his supervisor, who acted on behalf of the department chair, who acted on behalf of the university; all those agents are responsible in some way for the activity that took place but we do not say explicitly who bears responsibility and to what degree." As the quote shows, the delegation relation allows the provenance to record a chain of command.

The graphical and Prov-N representations of these relations is shown in Figure 5. Again the arrow points from the subject to the object, for example Entity1 was attributed to Alice. For the delegation relation, the position of the arrows pointing to both Alice and Acitivity2 from Bob mean that Bob acted on behalf of Alice to perform Activity 2. It is worth noting here that the delegation relation does not have to include an activity and could be used to show the general structure of the organisation.

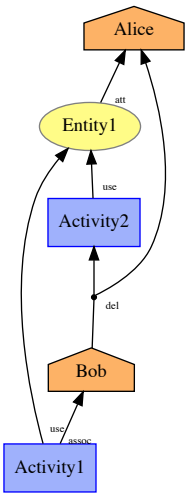


(a)

```
entity(Entity1)
entity(Entity2)

activity(Activity1)
activity(Activity2)

used(Activity1,Entity1,-)
used(Activity2,Entity2,-)

wasGeneratedBy(Entity2,Activity1,-)

wasDerivedFrom(Entity2, Entity1, Activity1,-,-)

wasInformedBy(Activity2,Activity1)
```

(b)

Figure 4: Key Prov relations between entities and activities in both graphic (a) and Prov-N (b) form.



(a)

```
entity(Entity1)

activity(Activity1)
activity(Activity2)

agent(Alice)
agent(Bob)

wasAttributedTo(Entity1,Alice)
wasAssociatedWith(Activity1,Bob,-)
actedOnBehalfOf(Bob,Alice,Activity2)

used(Activity1,Entity1,-)
used(Activity2,Entity1,-)
```

(b)

Figure 5: Key Prov relations between agents and entities/activities in both graphic (a) and Prov-N (b) form.

A.3 OSLC Specifications

The OSLC contains multiple specifications covering areas including requirements management, change management, asset management and others. This subsection is further divided into a series of sub-sub-sections, the first five (Appendices A.3.1 – A.3.5) compile lists of the main relations in each of these specifications along with a verbatim copy of the description of the relation taken from the OSLC specification documents. It is fair to say that the description text given by the OSLC specification for many of the relations is somewhat loose and open to some interpretation and so it is suggested that the reader does not dwell upon the detail of the descriptions in these subsections. The final subsubsection, Appendix A.3.6, contains a list of the relations that are currently selected for adoption for the INTO-CPS traceability ontology during the next year of the project, here we include our interpretation of the meaning of each adopted term.

Some of the specifications below make reference to the Dublin Core using the prefix 'dcterms:', this is an initiative to standardise many of the terms that are commonly used when identifying meta-data. A complete list of the meta-data terms they define may be found at their website[36].

A.3.1 Requirements Management (oslc_rm) v2.0

The OSLC requirements management specification considers the relations between requirements artefacts themselves and also between requirements and model elements that validate or satisfy them. The full specification may be found at the url in the footnote.[37].

oslc_rm:elaboratedBy The subject is elaborated by the object. For example, a collection of user requirements elaborates a business need, or a model elaborates a collection of system requirements.

oslc_rm:elaborates The object is elaborated by the subject.

oslc_rm:specifiedBy The subject is specified by the object. For example, a model element might make a requirement collection more precise.

oslc_rm:specifies The object is specified by the subject.

oslc_rm:affectedBy The subject is affected by the object, for example, a defect or issue.

oslc_rm:trackedBy Resource, such as a change request, which manages this requirement collection.

oslc_rm:implementedBy Resource, such as a change request, which implements this requirement collection.

oslc_rm:validatedBy Resource, such as a test plan, which validates this requirement collection.

[36] `http://dublincore.org/documents/dcmi-terms/`
[37] `http://open-services.net/bin/view/Main/RmSpecificationV2?rev=57`

oslc_rm:satisfiedBy The subject is satisfied by the object. For example, a collection of user requirements is satisfied by a requirement collection of system requirements.

oslc_rm:satisfies The object is satisfied by the subject.

oslc_rm:decomposedBy The subject is decomposed by the object. For example, a collection of business requirements is decomposed by a collection of user requirements.

oslc_rm:decomposes The object is decomposed by the subject.

oslc_rm:constrainedBy The subject is constrained by the object. For example, a requirement collection is constrained by a requirement collection.

oslc_rm:constrains The object is constrained by the subject.

A.3.2 Architecture Management (oslc_am) V3.0

The OSLC architecture management specification aims to represent the relations between architectural elements in general allowing the description of enterprise architectures, solution architectures and technical architectures. It has concepts that overlap with those in the requirements management specification. Only a small subset is presented below, the full specification may be found at the url in the footnote[38].

oslc_am:refines This resource is a refinement of the referenced resource. For example, a Use Case scenario might be a refinement of a textual requirement that describes the interaction.

oslc_am:satisfies This resource satisfies a requirement (the referenced resource). For example a UML Component satisfies a requirement to provide some type of functionality.

oslc_am:verifies A dependency from a model element to a requirement that determines whether a system fulfills the requirement. For example a Sequence diagram verifies a requirement that describes a protocol.

dm:derives The model element derives from a requirement.

dm:elaborates The model element elaborates a change request.

A.3.3 Asset Management (oslc_asset) V2.0

The OSLC asset management specification allows the relationships to an organisation's assets and the asset's lifecycle to be recorded. An asset is anything that provides value through reference or reuse and so this specification may be useful when recording the provenance of simulation results where hardware assts are used (HiL)[39].

[38] http://open-services.net/wiki/architecture-management/OSLC-Architecture-Management-Specification-2.0/

[39] http://open-services.net/wiki/asset-management/OSLC-Asset-Management-2.0-Specification/

oslc_asset:guid An identifier for the asset. Assigned by the service provider when a resource is created. Different versions of the same asset will share the same identifier.

oslc_asset:version The version of the asset. Possible values may include '1.0', '2.0', etc.

dcterms:abstract Short description or often a single line summary of the resource

dcterms:type The type of the asset based on values defined by the service provider. This specification does not define the resource for this property, however it should contain a dcterms:title property.

oslc_asset:state sed to indicate the state of the asset based on values defined by the service provider. This specification does not define the resource for this property, however it should contain a dcterms:title property.

oslc_asset:categorization A categorization to classify an asset. The category schema values are defined by the service provider. This specification does not define the resource for this property, however it should contain a dcterms:title property.

oslc_asset:manufacturer The name of the asset manufacturer.

oslc_asset:model The value of the asset model.

oslc_asset:serialNumber The serial number assigned by the asset manufacturer.

oslc_asset:tag Specifies the asset tag value for an Asset. Asset tags are typically human readable labels. For hardware assets, these tags are durable, securely attached to equipment, and may also be readable by barcode and/or RFID.

oslc_asset:artifact must reference an 'oslc_asset:Artifact' An Artifact fragment contained in this Asset resource.

oslc_asset:artifactFactory Resource URI used to post new artifacts to the asset.

dcterms:relation This relationship is loosely coupled and has no specific meaning. Details about this relationship may be included in a reified statement.

oslc_asset:relationshipType The type of this relationship from the perspective of the dcterms:relation resource based on values defined by the service provider. This specification does not define the resource for this property, however it should contain a dcterms:title property.

dcterms:creator Creator or creators of the relationship. It is likely that the target resource will be a foaf:Person but that is not necessarily the case.

dcterms:created Timestamp of the relationship creation.

dcterms:modified Timestamp of the latest relationship modification.

oslc_asset:state Used to indicate the state of the relationship based on values defined by the service provider. This specification does not define the resource for this property, however it should contain a dcterms:title property.

A.3.4 Change Management (oslc_cm) V3.0

The OSLC change management specification, as the name suggests is all about capturing and tracking change requests for a system. The relations it defines have not yet been included in the ontology but they may be in the future[40].

properties:

oslc_cm:closeDate The date at which no further activity or work is intended to be conducted.

oslc_cm:state Used to indicate the status of the change request. This property is read-only, but can be changed using Actions.

oslc_cm:action An action to change the state of this ChangeRequest.

oslc_cm:priority Priority of this ChangeRequest

oslc_cm:severity Severity or criticality of ChangeRequest

oslc_cm:attachment Multi-valued property of attachments associated with the Change Request.

relationship properties:

oslc_cm:related This relationship is loosely coupled and has no specific meaning. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_cm:affects Change request affects a plan item. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_cm:affectedByDefect Change request is affected by a reported defect. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_cm:tracksRequirement Tracks the associated Requirement or ChangeSet resources[41]. It is likely that the target resource will be an oslc_rm:Requirement but that is not necessarily the case.

oslc_cm:implementsRequirement Implements associated Requirement. It is likely that the target resource will be an oslc_rm:Requirement but that is not necessarily the case.

oslc_cm:affectsRequirement Change request affecting a Requirement. It is likely that the target resource will be an oslc_rm:Requirement but that is not necessarily the case.

oslc_cm:tracksChangeSet Tracks SCM change set resource. It is likely that the target resource will be an oslc_scm:ChangeSet but that is not necessarily the case.

[40] `http://open-services.net/wiki/change-management/Specification-3.0/`

[41] Change set resources are defined in the OSLC specification for software configuration management (oslc_scm) `http://open-services.net/bin/view/Main/ScmSpecV1`

A.3.5 Quality Management (oslc_qm) V2.0

The OSLC quality management specification focusses on the relations to system test plans, test cases and test results. These relations may prove to be valuable as the project works towards satisfying requirement 0092[42] on providing guidance for the production of argument evidence. Its full specification may be found at the url in the footnote[43].

oslc_qm:relatedChangeRequest A related change request. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_qm:usesTestCase Test Case used by the Test Plan. It is likely that the target resource will be an oslc_qm:TestCase but that is not necessarily the case.

oslc_qm:validatesRequirementCollection Requirement Collection that is validated by the Test Plan. It is likely that the target resource will be an oslc_rm:RequirementCollection but that is not necessarily the case.

oslc_qm:testsChangeRequest Request tested by the Test Case. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_qm:usesTestScript Test Script used by the Test Case. It is likely that the target resource will be an oslc_qm:TestScript but that is not necessarily the case.

oslc_qm:validatesRequirement Requirement that is validated by the Test Case. It is likely that the target resource will be an oslc_rm:Requirement but that is not necessarily the case.

oslc_qm:executionInstructions Instructions for executing the test script. Note that the value of Occurs is undefined. The resource shape document provided by the QM service provider may be consulted for its value.

oslc_qm:blockedByChangeRequest Change Request that prevents execution of the Test Execution Record. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_qm:runsOnTestEnvironment Indicates the environment details of the test case for this execution record.

oslc_qm:reportsOnTestPlan Test Plan that the Test Execution Record reports on. It is likely that the target resource will be an oslc_qm:TestPlan but that is not necessarily the case.

oslc_qm:runsTestCase Test Case run by the Test Execution Record. It is likely that the target resource will be an oslc_qm:TestCase but that is not necessarily the case.

oslc_qm:affectedByChange Change request that affects the Test Result. It is likely that the target resource will be an oslc_cm:ChangeRequest but that is not necessarily the case.

oslc_qm:executesTestScript Test Script executed to produce the Test Result. It is likely that the target resource will be an oslc_qm:TestScript but that is not

[42] See deliverable D7.3 [LPH+15] for further details on the requirements.
[43] `http://open-services.net/bin/view/Main/QmSpecificationV2`

necessarily the case.

oslc_qm:producedByTestExecutionRecord Test Execution Record that the Test Result was produced by. It is likely that the target resource will be an oslc_qm:TestExecutionRecord but that is not necessarily the case.

oslc_qm:reportsOnTestCase Test Case that the Test Result reports on. It is likely that the target resource will be an oslc_qm:TestCase but that is not necessarily the case.

oslc_qm:reportsOnTestPlan Test Plan that the Test Result reports on. It is likely that the target resource will be an oslc_qm:TestPlan but that is not necessarily the case.

A.3.6 Terms to adopt

Of the very many relationships defined in the previous specifications five have been selected as candidates for use in the INTO-CPS traceability ontology, these are:

oslc_rm:constrains The object is constrained by the subject. *Used to relate one requirement to another*

oslc_rm:decomposes The object is decomposed by the subject. *Used to related on requirement to another*

oslc_rm:satisfies The object is satisfied by the subject. *Used to relate an element in the architecture to a requirement it is intended to satisfy*

oslc_am:satisfies This resource satisfies a requirement (the referenced resource). For example a UML Component satisfies a requirement to provide some type of functionality. *Used to relate a simulation model or model check model to a requirement it is intended to satisfy*

oslc_am:verifies A dependency from a model element to a requirement that determines whether a system fulfills the requirement. For example a Sequence diagram verifies a requirement that describes a protocol. *Used to relate a simulation, DSE or model check result to a requirement. It represents evidence for that requirement being met.*

One notable omission from the OSLC specifications was a relation that would permit us to link a simulation result to a requirement to indicate that the requirement was not met. It could be considered that if the oslc_am:verifies link is not used then the result does not support a requirement being met, but it is appealing to be able to make this explicit. In response to this some of the views contain a new relation that is currently termed `into:doesNoVerify`. This relation means the complete opposite to the oslc_am:verifies, and would say that the simulation, DSE or model check result shows that the design does not meet the requirement in question.

It should be noted that while only a subset of the specification relations have been used in the following ontology, this does not rule out their use, as the ontology derives from the workflow components that have been explored so far and so the set of relations is expected to increase in the future.

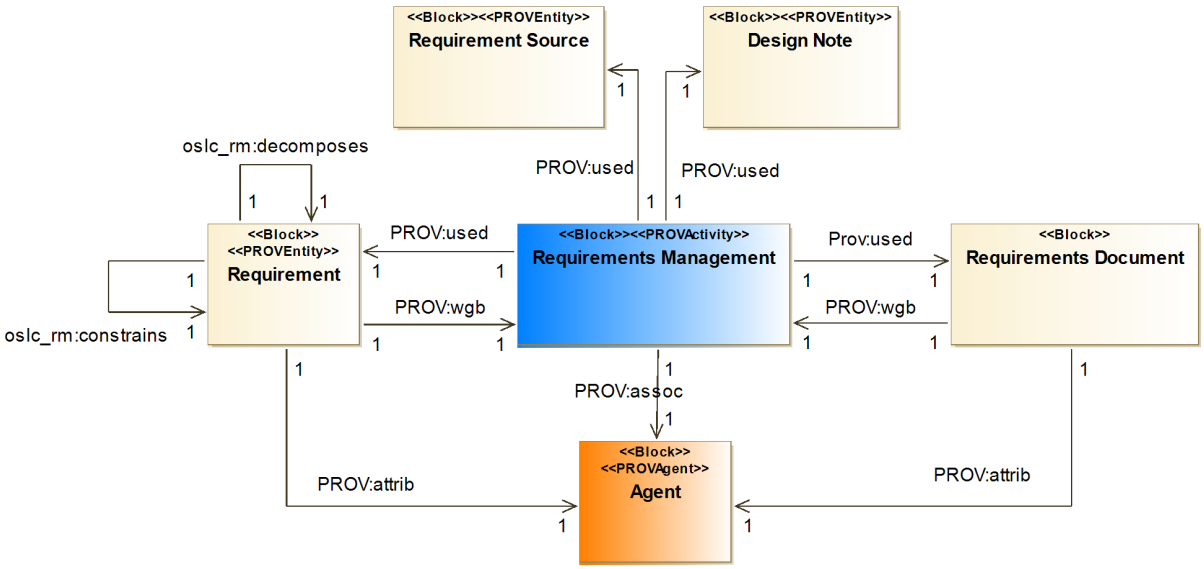


Figure 6: Block Definition Diagram (BDD) of the requirements activity

A.4 Proposed INTO-CPS Traceability Ontology

In subsection A.2 the concepts of PROV were introduced and in subsection A.3 we outlined the pertinent OSLC specification with some discussion about which are likely useful for the INTO-CPS traceability work. In this subsection we will combine both the PROV and OSLC work to describe an ontology that will form the basis of the provenance and traceability data in INTO-CPS.

The ontology is presented as a collection a views where the majority of the views are centred around one or more activities that will take place while using the INTO-CPS tool chain. As with the PROV figures earlier a blue element represent an activity, a yellow element represents an entity and an orange element represents an agent.

A.4.1 Requirements

Starting with requirements, Figure 6 shows the activity of requirements management. This activity makes use of *design notes* and *requirement sources*, which are the primary documents from the stakeholders and it produces *requirements*. Note that the requirement has two OSLC relations to itself, these are to support recording of relationships between individual requirements. There are only two relations shown linking requirements to other requirements, however it is suggested that others from the OSLC requirements management specification also be allowed. Figure 7, shows that we expect there to be *requirements documents* that will contain one or more *requirements*.

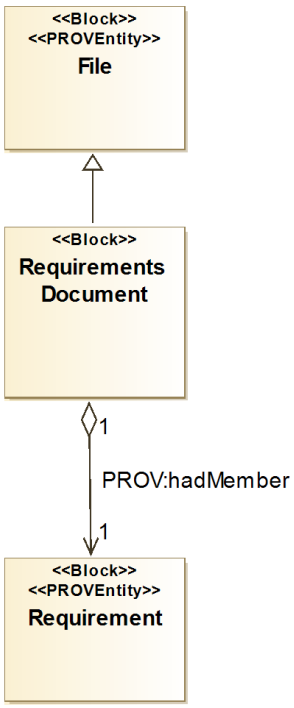


Figure 7: BDD of the requirements files

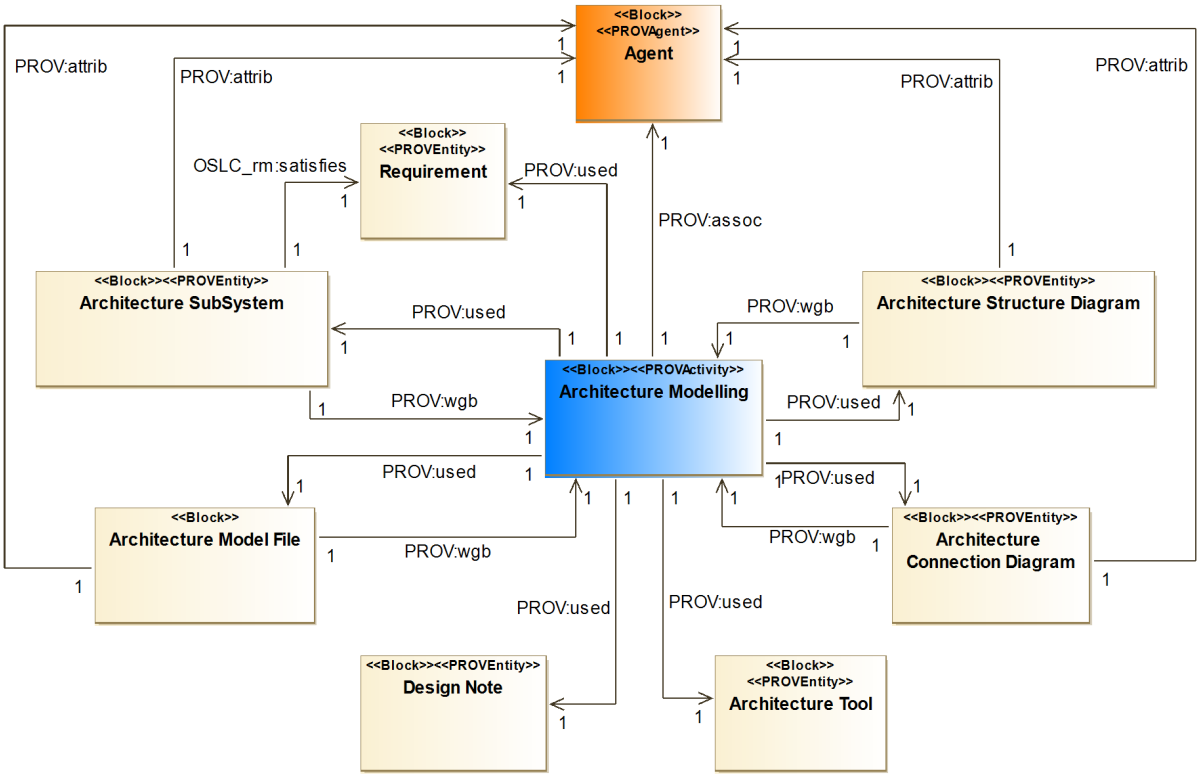


Figure 8: BDD Architecture modelling

A.4.2 Architecture Modelling

The activity of *architecture modelling* is presented in Figure 8. Architecture modelling is influenced by requirements, design notes and also previous version of its outputs, it produces the two views defined in the INTO-CPS SysML profile (*architecture structure diagram* and *architecture connection diagram*, described in deliverable D2.1a [APCB15]) and the *architecture subsystems* they contain. The architecture subsystems may be related to any requirements they satisfy. The files that that represent the architecture are shown in Figure 9.

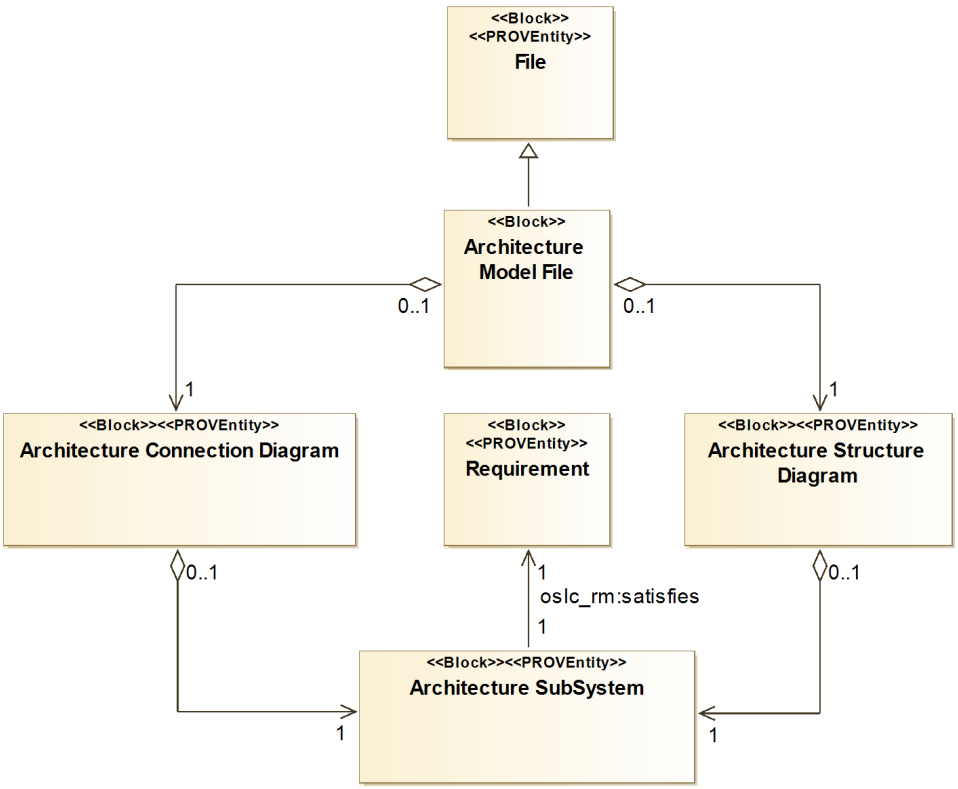


Figure 9: BDD Architecture modelling elements

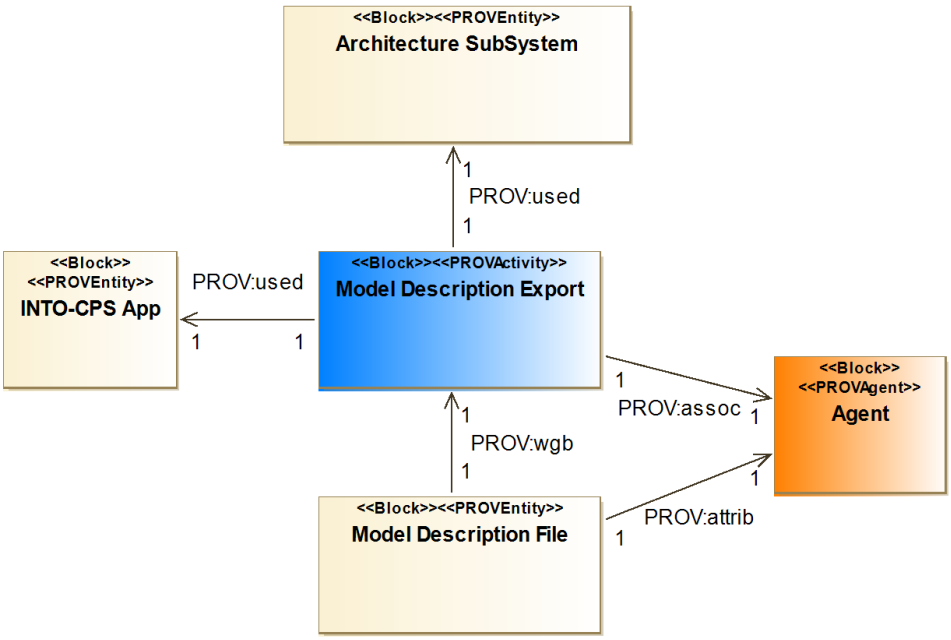


Figure 10: BDD of the model description export activity

A.4.3 Model Description File Export

The connection between the architecture and the simulation and model checking models is provided by the *model description file* and the *model description export* is presented in Figure 10. This functionality is provided by the *INTO-CPS app*.

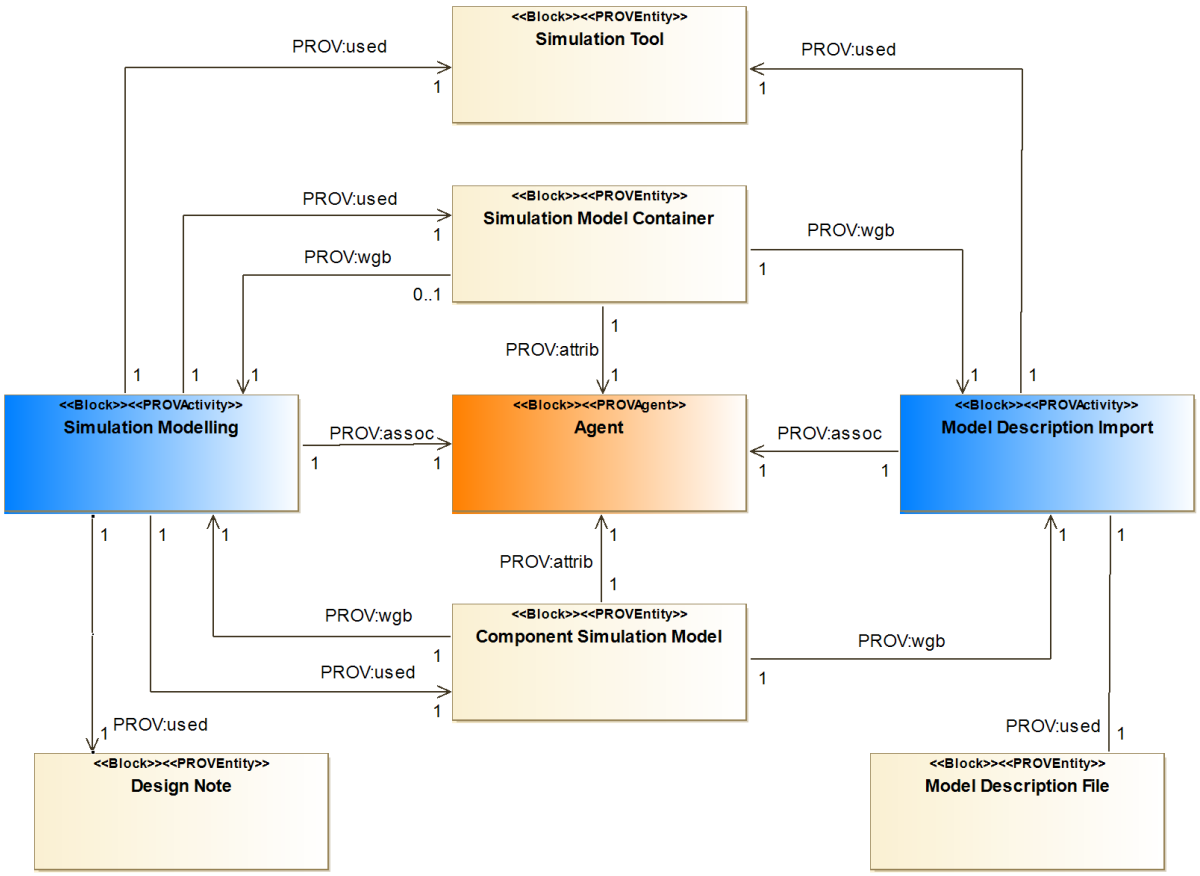


Figure 11: BDD showing simulation model creation activities

A.4.4 Simulation Models

There are two distinct activities shown in Figure 11, *model description import*, which creates a skeleton model in the chosen simulation tool, and *simulation modelling*, which represents the population of the model to do something useful. The output of both of these activities are the *component simulation model* and *simulation model container*. Figure 12 shows the file elements involved, where the simulation model container, for example a 20-sim .emx file, contains one or more component simulation models. The component simulation models may be linked to any requirements they satisfy via the *OSLC_ am:satsifies* relation.

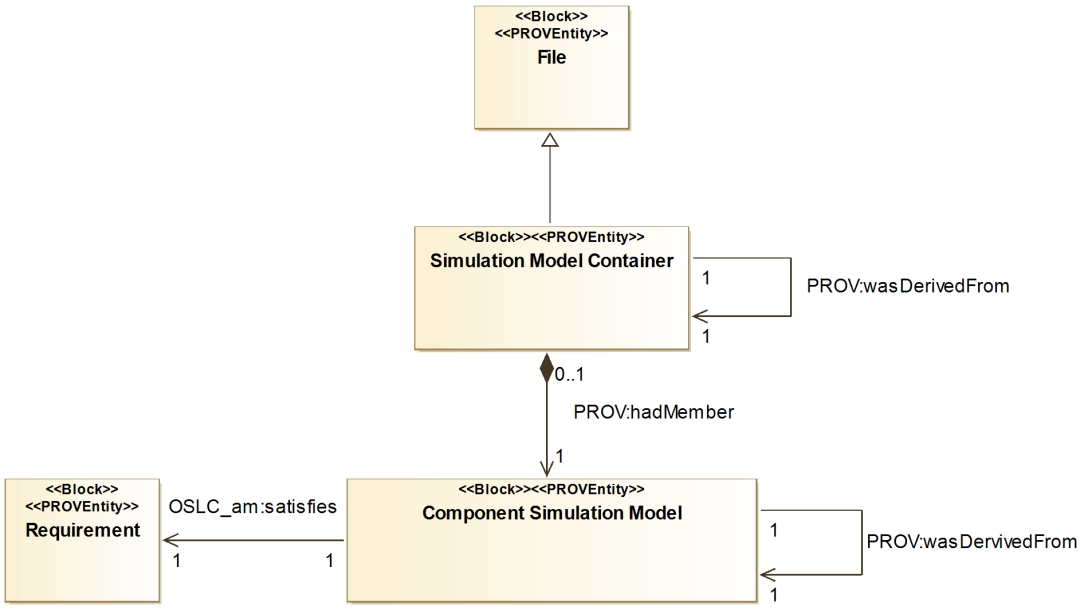


Figure 12: BDD showing simulation model files

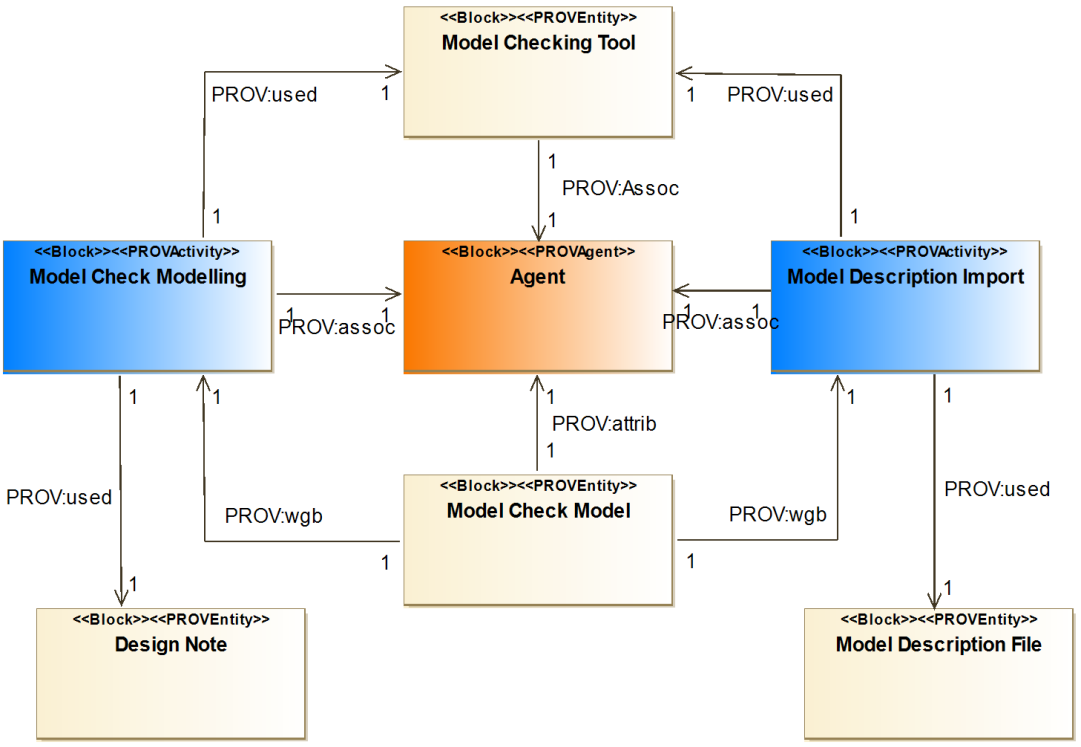


Figure 13: BDD showing model check model creation

A.4.5 Model Checking

Model checking begins with an identical structure model description import and *model check modelling*, Figure 13 as we saw for simulation models previously. Figure 14 shows the structure around both the *model checking* and *model check test creation* activities, these respectively output *model check results* and *model check test case*. The model check test result is the first time we have seen evidence that models meet or do not meet the specification, thus we see that it may connect via the *OSLC_ am:verifies* or *into:doesNotVerify* relations. The relationships between the files involved in model checking are shown in Figure 15

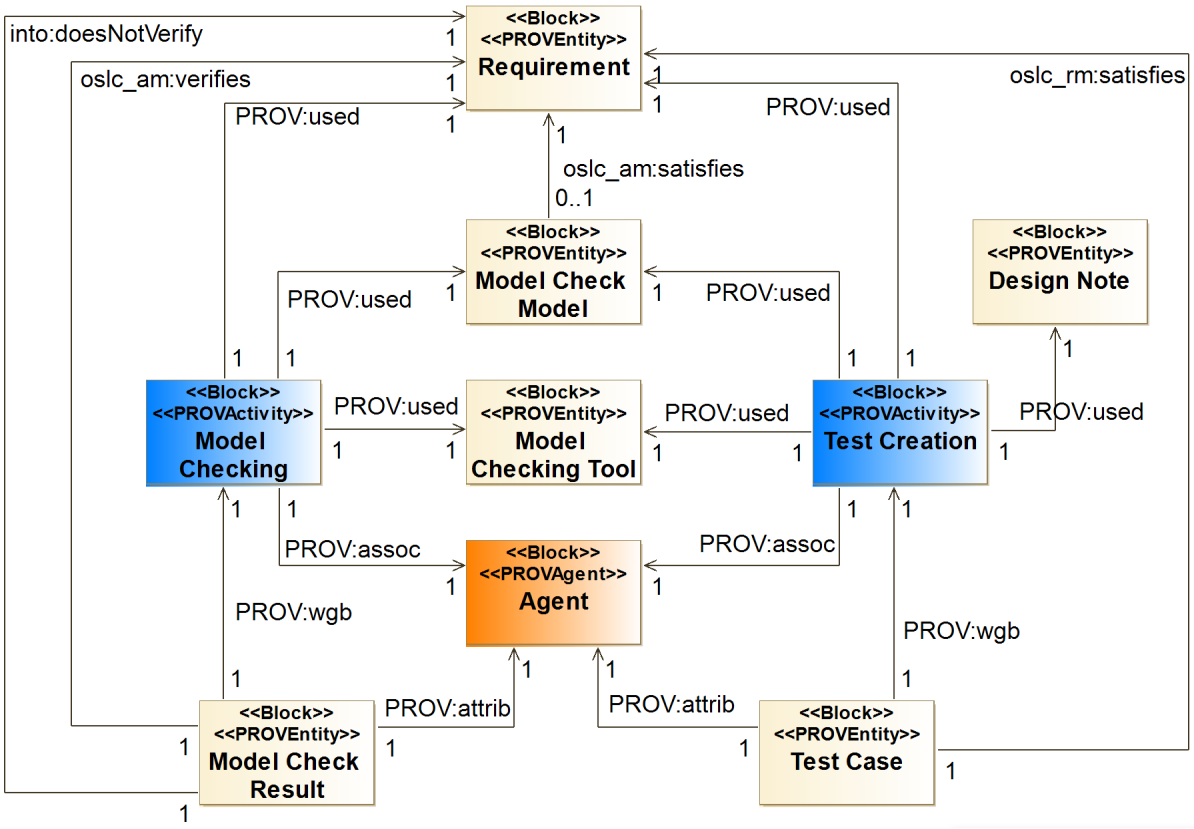


Figure 14: BDD showing the activities of model checking

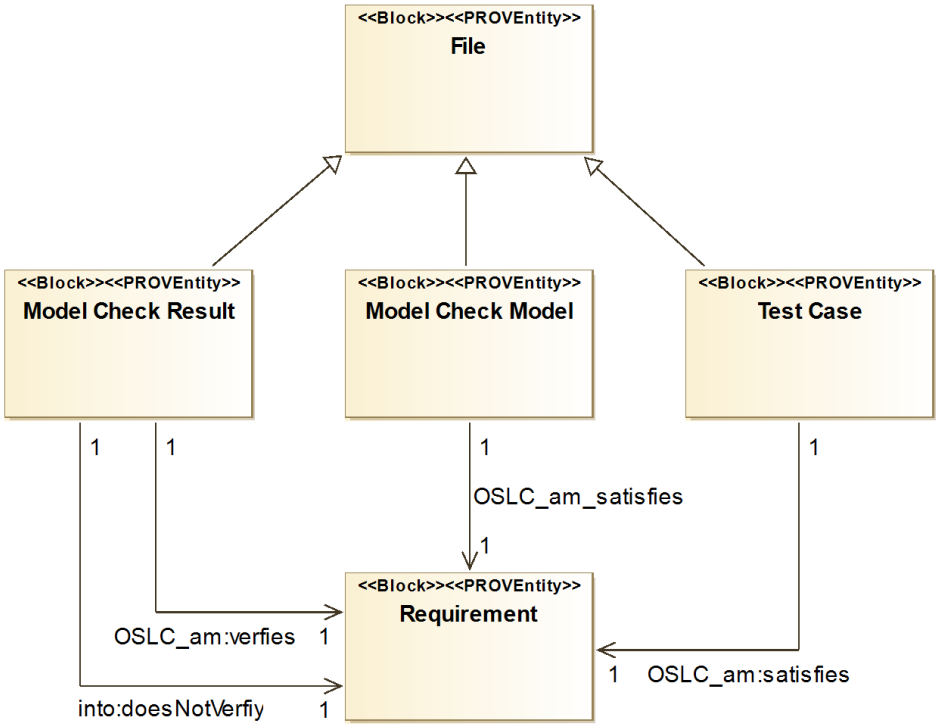


Figure 15: BDD showing the file elements around model checking

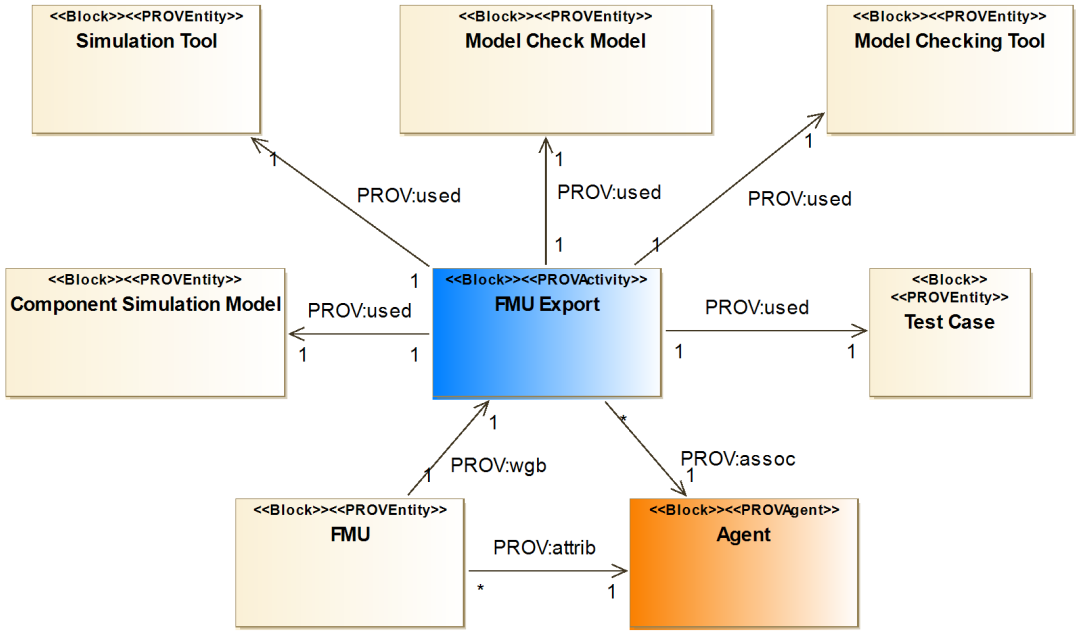


Figure 16: BDD showing simulation FMU generation

A.4.6 FMU and Code Generation

The generation of Functional Mockup Units (FMUs) and compilable source code is necessary before any simulation may take place in the INTO-CPS tool chain. Figure 16 presents the elements surrounding the *FMU export* activity, these are the simulation models, model check models and test cases from which FMUs can be generated and the tools used to generate them. The output is the FMU itself.

The creation of an FMU to support Hardware in the loop (HiL) simulation differs slightly in that instead of being generated from an simulation or model check model it will take a model description file as its input, Figure 17. Associated with this activity is a software agent that actually configures and compiles the FMU to permit communication with the hardware asset.

The activity of *code generation* is shown in Figure 18, where its inputs are simulation and model check models and its output is some form of *source code*. The files associated with code generation are presented in Figure 19.

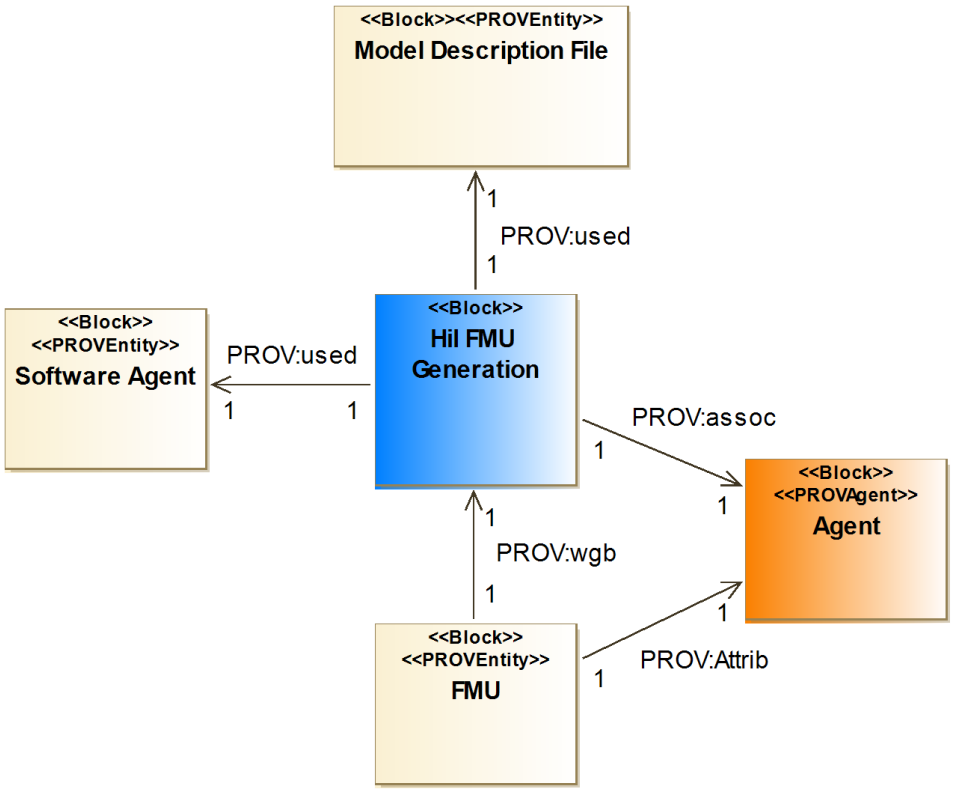


Figure 17: BDD showing FMU generation to allow HiL simulation

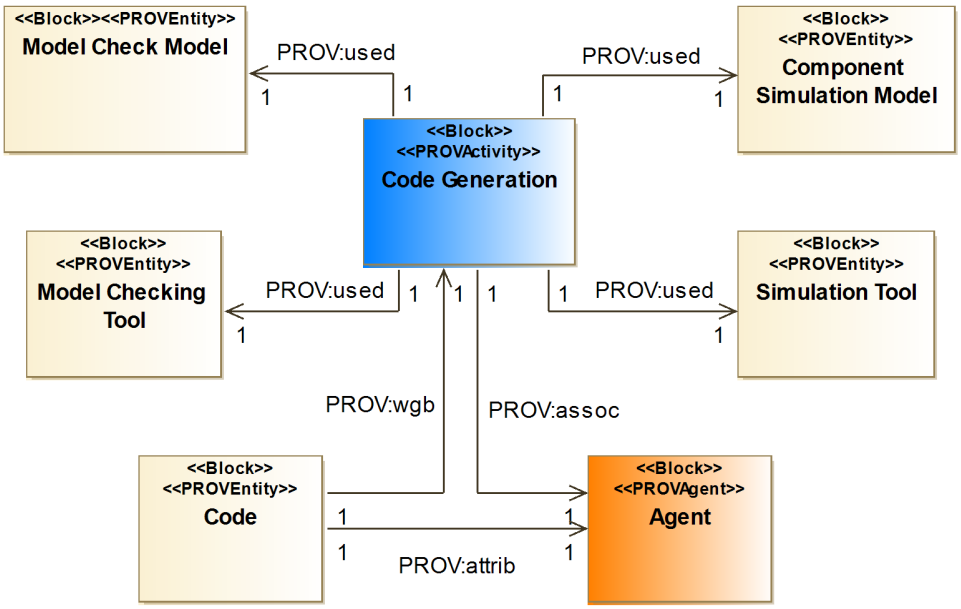


Figure 18: BDD Showing the activity of code generation

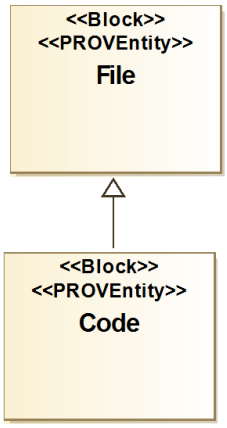


Figure 19: BDD showing the files associated with code generation

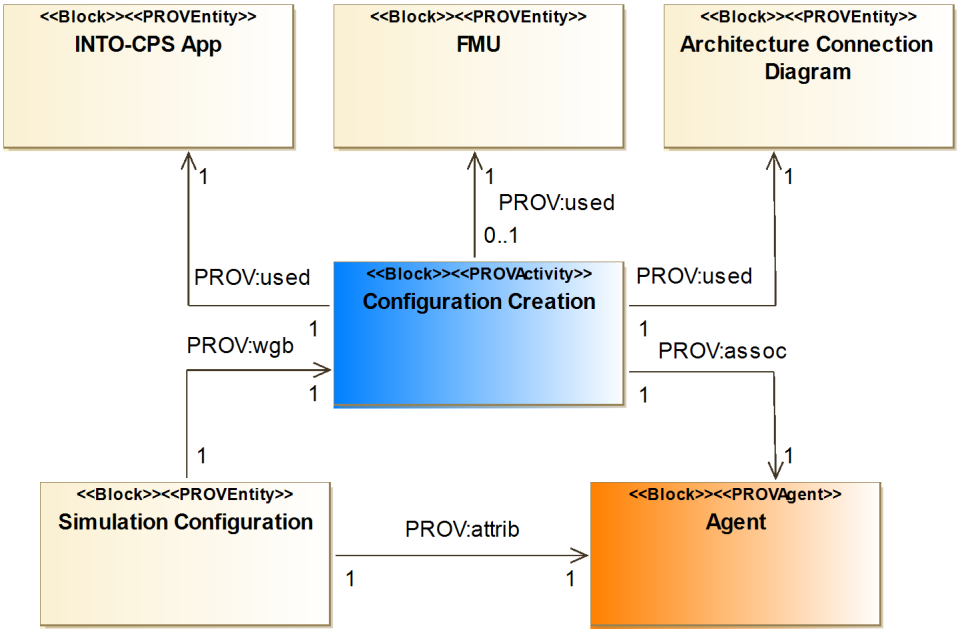


Figure 20: BDD showing the simulation configuration activity

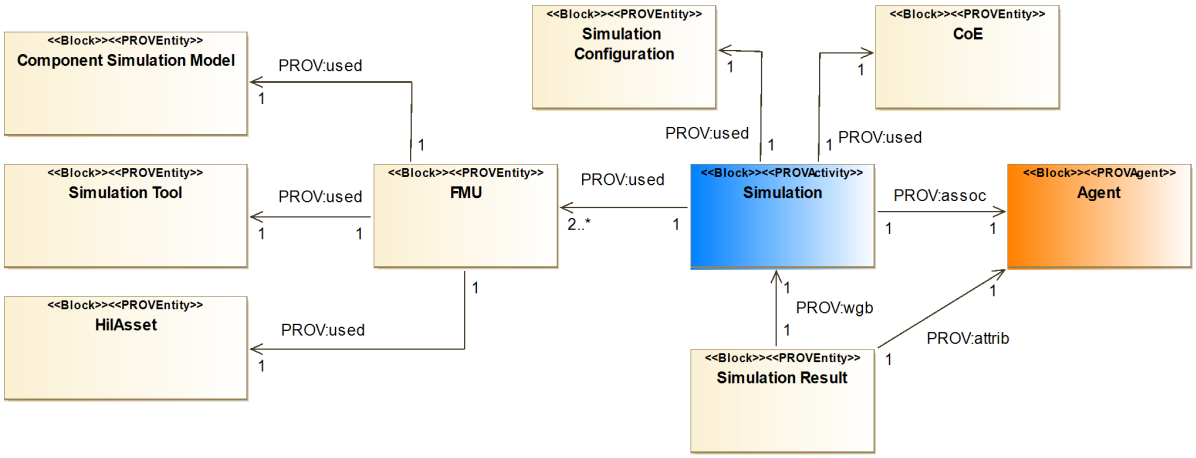


Figure 21: BDD showing the activity of simulaition

A.4.7 Simulation

Before simulation can take place we must produce a simulation configuration file, this activity is shown in Figure 20. Here the INTO-CPS app uses the generated FMUs and the architecture connection diagram to produce a *simulation configuration* which the co-simulation orchestration engine (*COE*) uses to execute a simulation.

Figure 21 shows the activity of *simulation* itself. Here we see that it makes use of the COE, simulation configuration and the FMUs that have already been generated. Here we also see that the FMUs themselves may make use of simulation models and simulation tools if the FMU is a wrapper or may reference a hardware asset if it is a HiL FMU.

Finally Figure 22 shows the files associated with simulation. Of note here is that the simulation result may be linked to a requirement is it provides evidence for or against that requirement being met.

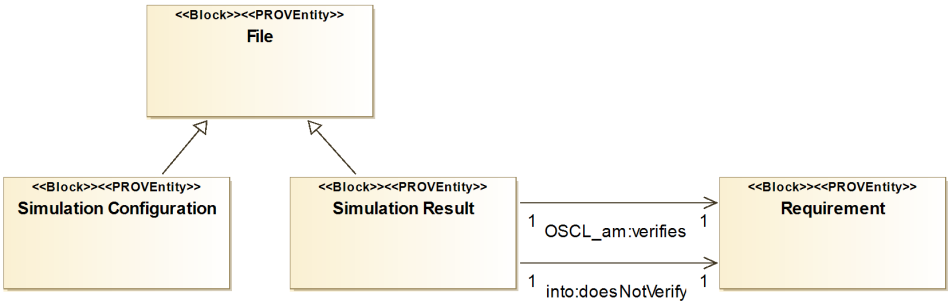


Figure 22: BDD showing the files associated with simulation

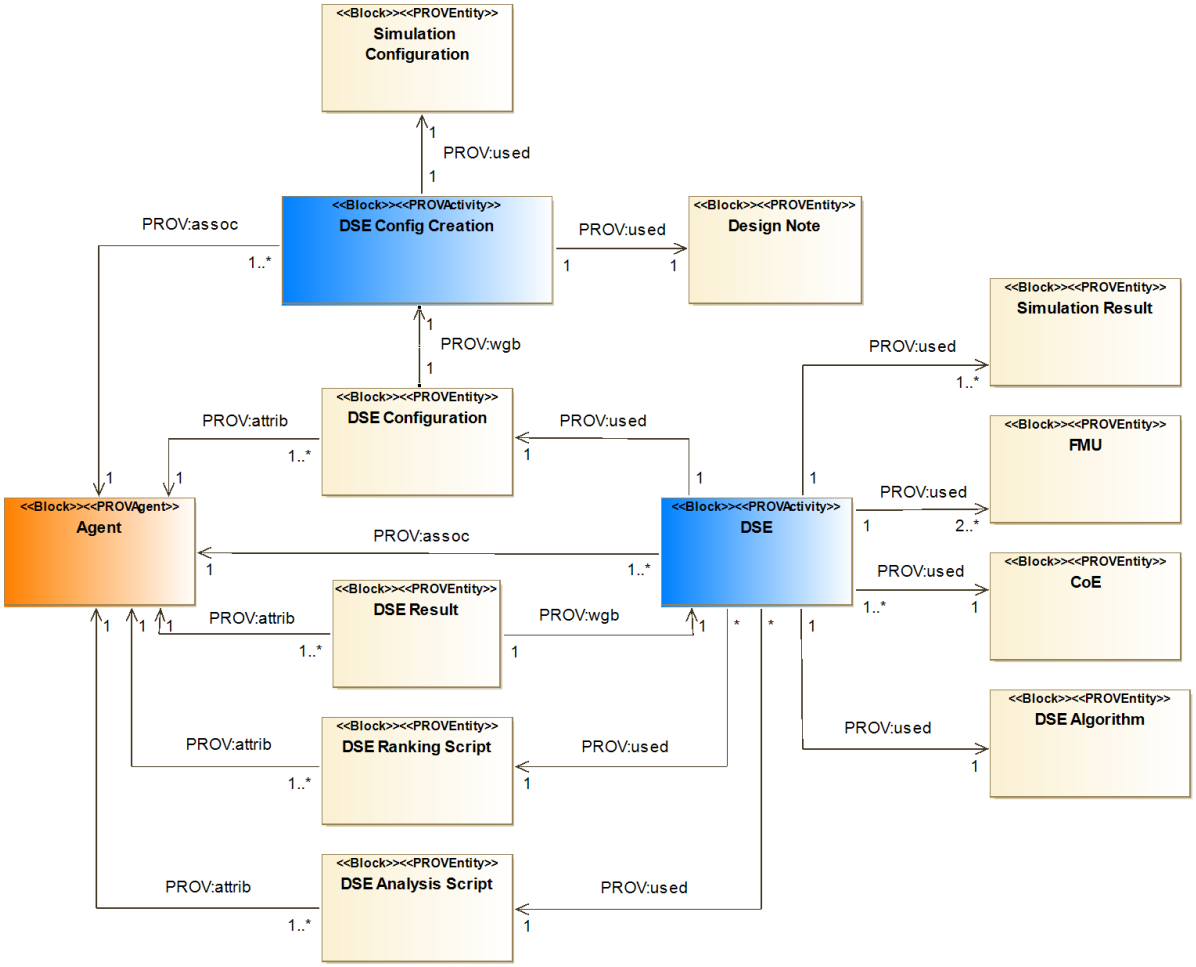


Figure 23: BDD showing the DSE activites

A.4.8 DSE

Figure 23 shows the two activities associated with DSE, the *DSE configuration creation* and *DSE* itself. The activity of DSE configuration takes a simulation configuration and queries the user to define the range of parameters and algorithm choices for the actual DSE itself. The DSE then uses the simulation configuration to select the appropriate scripts to launch simulations, evaluate the objective values of each simulation and then rank them as a result.

Finally in DSE, Figure 24 shows the files associated with these activities. Again we note that the DSE result, which is here shown as a single file but is likely not to be, may be linked to requirements as either evidence for or against it being met.

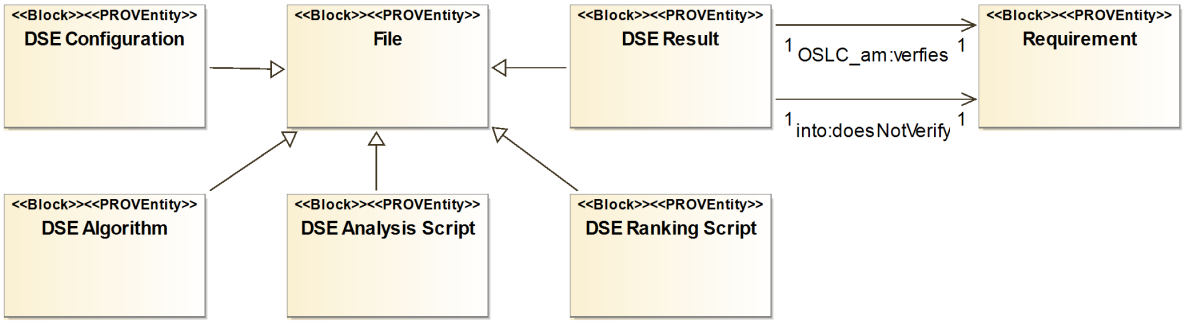


Figure 24: BDD showing the DSE files

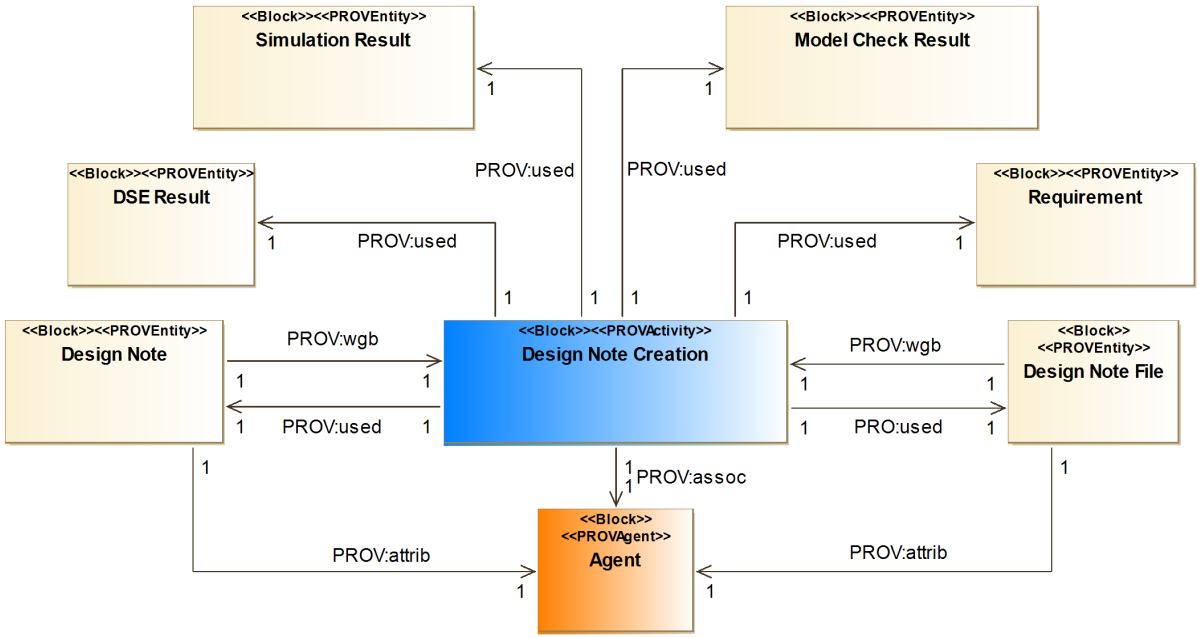


Figure 25: BDD showing design note creation

A.4.9 Design Notes

Throughout the previous views on the ontology many of the activities have made use of a *design note*. Here a design note may be any document that records rationale, decisions and directions for the modelling process. These may be produced at any point and are a vital part of understanding why the final product of a design process takes the form it does. In Figure 25 we see that the activity of design note creation may be linked to the outputs of many of the main activities in the INTO-CPS tool chain, this is important so we may revisit the evidence from which the note was created. Figure 26 shows the files associated with design notes.

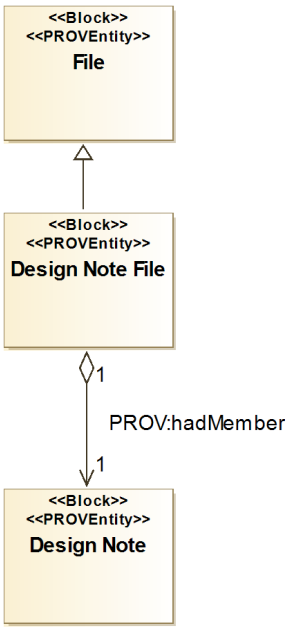


Figure 26: BDD showing the design note files

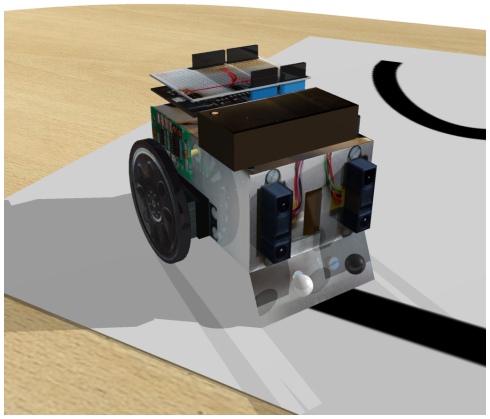


Figure 27: 3D view of the line follow robot generated by its co-model.

A.5 Robot Example

The robot example presented is inspired by the work that took place to produce the line following robot that proved very useful in the DESTECS[44] project, Figure 27. In the project there were many versions of the robot's constituent models generated and stored in multiple locations, so its real provenance graph is much larger than would be needed to demonstrate a potential use of PROV, thus we will present a greatly simplified version of the events here. We will also take two steps into the future and use a planned evolution of the model to represent how PROV may represent a simulation using INTO-CPS tooling.

The scenario starts with the with the simultaneous development of two models of the robot, a 20-sim CT model produced at University Twente (UT) and an Overture DE model produced by Ken Pierce at Newcastle University (UNEW). Both of these models included the body and controller of the robot but were lacking the line following sensors, so the first modification was for UNEW to take the UT 20-sim mode and add these sensors. With the sensors in place it was possible to integrate the 20-sim and overture models, add a contract and debug launch configuration to construct the first co-model, this was done by Ken at UNEW. The next step was to modify the model to experiment with the Design Space Exploration (DSE) capabilities of the Crescendo tool, so the co-model created by Ken was modified with an Automated Co-model Analysis (ACA) launch file added and the set of simulations run.

The shortened history of the line follow robot ends with the ACA run, the following steps outline the short term plan for converting the model for use with the INTO-CPS tooling. The first step in this process will be to dissect the existing 20-sim model into three model fragments containing the sensor, the body and wheels and finally the 3D view of the robot for visualisation. The sensor model and body and wheel models will then be used as a specification from which two equivalent open modelica models will be built. The final step in this process will be the construction of an INTO-CPS multi-model using a selection of the model fragments and the co-simulation orchestration engine (COE).

[44] `www.destecs.org`

In the following sub-subsections examples of how PROV could be used to record the events will be shown.

A.5.1 Model Introduction

There are examples of two types of model introduction in the robot study. On the right hand side of Figure 28(a) we can see the introduction of the CT model developed by UT. This model was supplied in its final form by UT and so we only need to record the identity of the entity (file) provided, the agent responsible for the entity and link them with a 'was attributed to' relation. This pattern could be used to represent any model that is provided by an external agency, for example if a tyre supplier provided a vehicle manufacturer with a compiled FMU for their tyre for use in simulations.

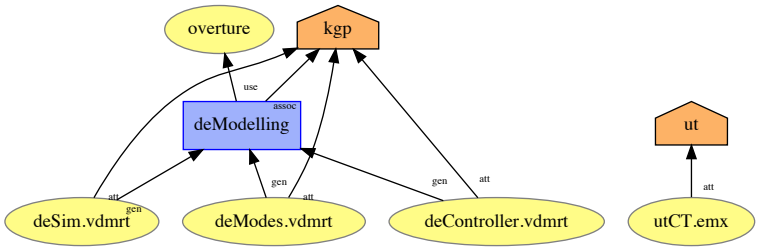
The structure on the left hand side of Figure 28(a) represents the internal development of new models. Here we record the identities of the model files produced, the name of the engineer responsible and the name and version of the tooling used. Central to the production of these models is the deModelling activity, apart from the id of activity the element can be used to record the start and/or end times of the modelling activity. The engineer, Ken, is associated with both the activity of modelling via a 'was associated with' relation and with the generated files via the 'was attributed to' relation. The files are also linked to the activity via the 'was generated by' relation. Finally we indicate which tool was used by connecting deModelling to Overture via a "used" relation.

A.5.2 Model Evolution

The PROV structure used to represent a new model entity being created from an existing one is very similar to that used for model introduction but with the addition of further "used" entities defining the which model files were used a sources. Figure 29 shows the evolution of the UT CT model to have line follow sensors. Note that utCT.emx is not only connected to the activity by the used relation but is also connected to the unewCT.emx entity via a *was derived from* relation. In this example there was only one source model and one generated model and so there is no ambiguity regarding which models influenced which other models, however this is not always the case. If we consider the two modelling exercises shown in Figure 30 we see that ambiguity could be present. In the modelling2 activity, where the single 20-sim model is decomposed into smaller components, we can see that there is one source model and two generated models and we can infer that the two generated models were both influenced in some way by the single source model. However, in the modelling3 activity, in which openModelica versions of the 20-sim component models are produced, there are two source models and two output models. Here the derived relation helps to make explicit which of the produced models was influenced by which of the source models. This explicit lineage could be important if there is a need to trace some property through the model chain.

A.5.3 Simulation

Figure 31 shows the PROV structure for the initial DE only simulation performed at UNEW. Here we list the input modes, the activity of simulation, the agent responsible

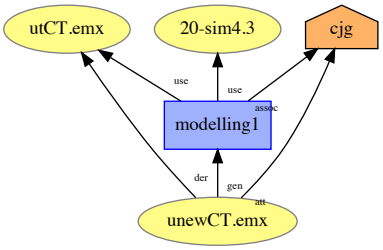


(a)

```
// Tools
entity(overture)

// UT CT model
entity(utCT.emx)
wasAttributedTo(utCT.emx,ut)

// UNEW DE model
activity(deModelling)
used(deModelling, overture,-)
wasAssociatedWith(deModelling, kgp,-)
wasGeneratedBy(deSim.vdmrt, deModelling,-)
wasGeneratedBy(deModes.vdmrt, deModelling,-)
wasGeneratedBy(deController.vdmrt, deModelling, -)
entity(deSim.vdmrt)
entity(deModes.vdmrt)
entity(deController.vdmrt)
wasAttributedTo(deSim.vdmrt,kgp)
wasAttributedTo(deModes.vdmrt,kgp)
wasAttributedTo(deController.vdmrt,kgp)
```

(b)

Figure 28: Prov representation of model introduction in both graphic (a) and Prov-N (b) form.

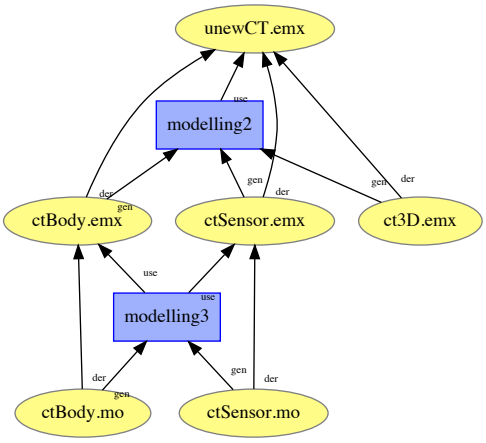


(a)

```
// Tools
entity(20-sim4.3)

// UT CT model
entity(utCT.emx)

// UNEW CT Sensors added and tested
entity(unewCT.emx)
wasAttributedTo(unewCT.emx,cjg)
activity(modelling1)
used(modelling1, 20-sim4.3,-)
wasGeneratedBy(unewCT.emx, modelling1,-)
wasAssociatedWith(modelling1, cjg,-)
used(modelling1,utCT.emx,-)
wasDerivedFrom(unewCT.emx,utCT.emx)
```

(b)

Figure 29: Prov representation of simple model evolution in both graphic (a) and Prov-N (b) form.

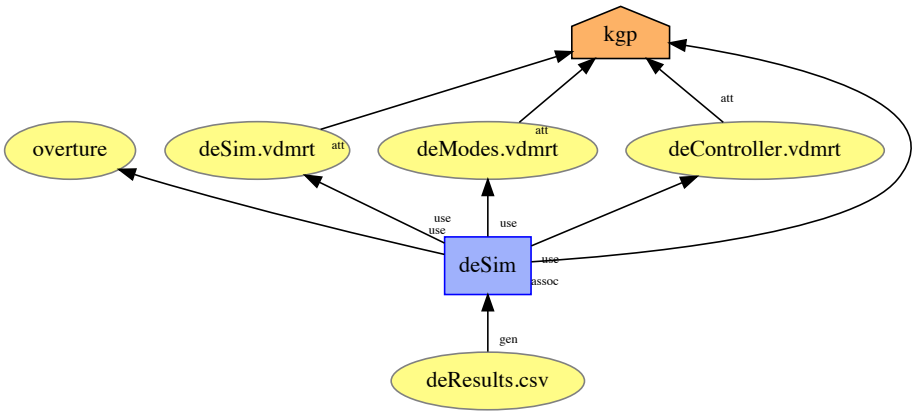


(a)

```
entity(unewCT.emx)

// creation of ct model fragments (20-sim)
activity(modelling2)
used(modelling2, unewCT.emx, -)
entity(ctBody.emx)
entity(ctSensor.emx)
entity(ct3D.emx)
wasGeneratedBy(ctBody.emx, modelling2, -)
wasGeneratedBy(ctSensor.emx, modelling2, -)
wasGeneratedBy(ct3D.emx, modelling2, -)
wasDerivedFrom(ctBody.emx, unewCT.emx)
wasDerivedFrom(ctSensor.emx, unewCT.emx)
wasDerivedFrom(ct3D.emx, unewCT.emx)

// creation of ct model fragments (open modelica)
activity(modelling3)
used(modelling3, ctSensor.emx, -)
used(modelling3, ctBody.emx, -)
entity(ctBody.mo)
entity(ctSensor.mo)
wasGeneratedBy(ctBody.mo, modelling3, -)
wasGeneratedBy(ctSensor.mo, modelling3, -)
wasDerivedFrom(ctBody.mo, ctBody.emx)
wasDerivedFrom(ctSensor.mo, ctSensor.emx)
```

(b)

Figure 30: Prov represel evolution in both graphic (a) and Prov-N (b) form.



(a)

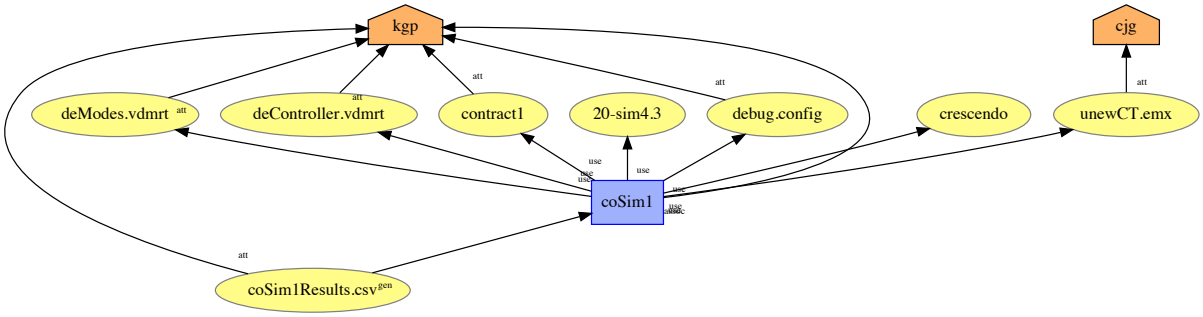
```
// agents
agent(kgp)

// Tools
entity(overture)

// UNEW DE model
entity(deSim.vdmrt)
entity(deModes.vdmrt)
entity(deController.vdmrt)
wasAttributedTo(deSim.vdmrt,kgp)
wasAttributedTo(deModes.vdmrt,kgp)
wasAttributedTo(deController.vdmrt,kgp)
activity(deSim)
wasAssociatedWith(deSim, kgp, -)
used(deSim, deSim.vdmrt,-)
used(deSim, deModes.vdmrt,-)
used(deSim, deController.vdmrt,-)
used(deSim, overture,-)
entity(deResults.csv)
wasGeneratedBy(deResults.csv, deSim,-)
```

(b)

Figure 31: Prov represel evolution in both graphic (a) and Prov-N (b) form.

along with any results files generated. The link to the simulation tool is important as this defines the tool, it's version and potentially also the ID of the host upon which the tool was running, the benefit of this detail is that if there is an issue discovered with either a specific tool version or an individual installation then the simulation results that it produced may be identified and examined or re-run.

Co-simulation using the Crescendo tool is shown in Figure 32. Again this includes the input models, output results and the responsible agent. It differs slightly from the DE simulation as now we have to represent two tools instead of just one and also we have to track the auxiliary files that are required to completely describe the co-simulation, these are the contract and debug.config. In the case that the co-simulation used the DSE feature of the Crescendo then there will by multiple instances of coSim1Results.csv, at least one file per simulation run and each of these will need to be included in the provenance individually.



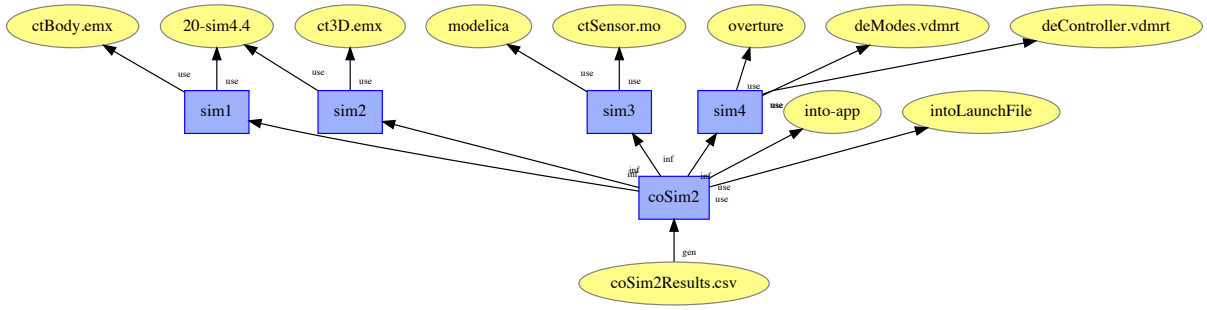
(a)

```
agent(cjg)
agent(kgp)

// Tools
entity(20-sim4.3)
entity(crescendo)

// UNEW DE model
entity(deModes.vdmrt)
entity(deController.vdmrt)
wasAttributedTo(deModes.vdmrt,kgp)
wasAttributedTo(deController.vdmrt,kgp)

// UNEW CT Sensors added and tested
entity(unewCT.emx)
wasAttributedTo(unewCT.emx,cjg)

// Initial crescendo co-model integration and test
entity(contract1)
entity(debug.config)
wasAttributedTo(contract1, kgp)
wasAttributedTo(debug.config, kgp)
activity(coSim1)
wasAssociatedWith(coSim1, kgp, -)
used(coSim1, contract1,-)
used(coSim1, deController.vdmrt,-)
used(coSim1, deModes.vdmrt,-)
used(coSim1, unewCT.emx,-)
used(coSim1, debug.config, -)
used(coSim1, crescendo, -)
used(coSim1, 20-sim4.3,-)
entity(coSim1Results.csv)
wasAttributedTo(coSim1Results.csv,kgp)
wasGeneratedBy(coSim1Results.csv,coSim1,-)
```

(b)

Figure 32: Prov represel evolution in both graphic (a) and Prov-N (b) form.

For the INTO-CPS style of co-simulation we need to use a slightly different structure to properly represent the configuration of the co-model and to allow replay, such an envisaged structure is show in Figure 33. This structure still contains the main elements of the previous simulations, the input models, tools, agents and auxillary files, but there is now an additional layer of activities between the input models and the co-simulation activity. For example, ctBody.emx is *used* by sim1 and coSim2 is *informed* by sim1, similarly ct3D.emx is *used* by sim2 and coSim2 is *informed* by sim2. The reason behind this structure is that we may choose to have multiple simulations that use the same tool family, e.g. 20-sim, but due to the distributed nature of the COE could be executed on different instances the tool, or different versions on different hosts. So in the figure we use simX to allow use to be more explicit about which models were running on which tools, where the tools can also specify the host and version. This structure also closely mimics the structure of an INTO-CPS co-simulation, with each simX representing one FMU and the *informed* relations representing the data shared over the FMI bus.



(a)

```
// Tools
entity(20-sim4.4)
entity(overture)
entity(into-app)
entity(modelica)

// UNEW DE model
entity(deModes.vdmrt)
entity(deController.vdmrt)

// creation of ct model fragments (20-sim)
entity(ctBody.emx)
entity(ct3D.emx)

// creation of ct model fragments (open modelica)
entity(ctSensor.mo)

// construction of into co-model and simulation
activity(sim1)
used(sim1, ctBody.emx, -)
used(sim1, 20-sim4.4, -)
activity(sim2)
used(sim2, ct3D.emx, -)
used(sim2, 20-sim4.4, -)
activity(sim3)
used(sim3, ctSensor.mo, -)
used(sim3, modelica, -)
activity(sim4)
used(sim4, deModes.vdmrt, -)
used(sim4, deController.vdmrt, -)
used(sim4, overture, -)
entity(intoLaunchFile)
activity(coSim2)
used(coSim2, intoLaunchFile, -)
used(coSim2, into-app, -)
wasInformedBy(coSim2, sim1)
wasInformedBy(coSim2, sim2)
wasInformedBy(coSim2, sim3)
wasInformedBy(coSim2, sim4)
entity(coSim2Results.csv)
wasGeneratedBy(coSim2Results.csv, coSim2, -)
```

(b)

Figure 33: Prov represel evolution in both graphic (a) and Prov-N (b) form.

A.5.4 Requirements and Submodels

To outline how requirements can be traced we start by assuming the line following robot as having two main requirements, requirement R1, that it can move and requirements R2, that it can detect lines to follow. The first step in this example is the construction of the SysML model of the robot, the resulting model, which is described in deliverable D3.4 [FGP+15] is shown along with the two requirements in Figure 34. There are two submodels in this model that the modeller has determined contribute to the satisfaction of R1 and R2 and as such two links are created connecting these submodels to the requirements. The links use the `oslc_rm:satisfies` (shortened to 'sat' in the figure) relation to indicate that the require is, at least in part, satisfied by the part of the CPS the submodel describes.

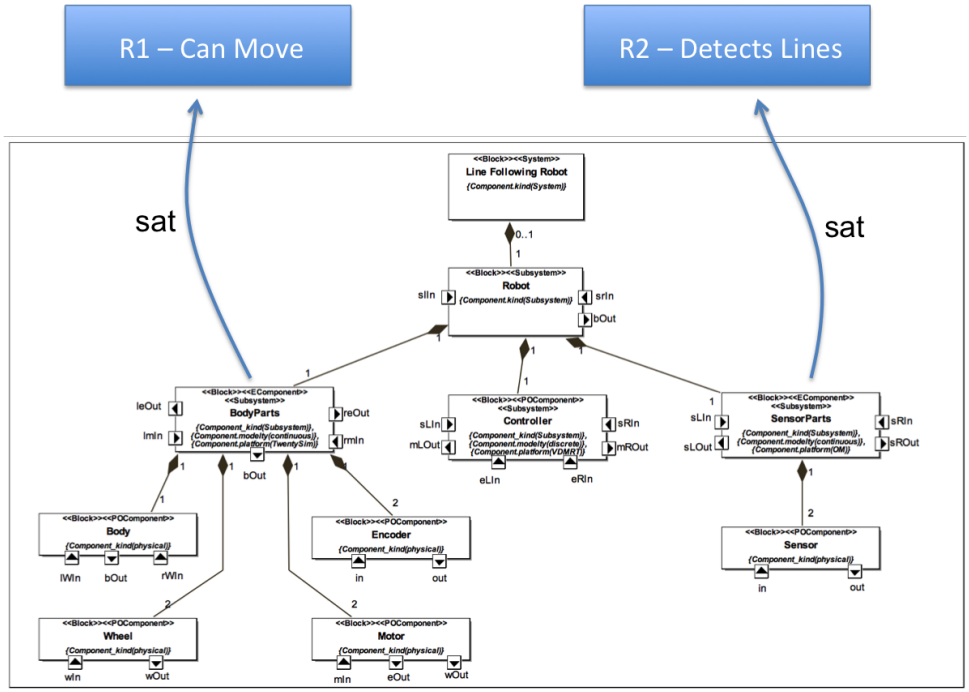


Figure 34: The intial SysML model of the line follow robot with links to requirements. (The text is in the SysML blocks is not important.)

The next step in the process is to export the model description files and use them to create simulation models. For the purpose of this example, let us imagine that two teams produce independent 20-sim models of the robot and these models, `a.emx` and `b.emx`, Figure 35. When these models are built and committed, both teams link their submodels to the requirements they implement using the `oslc_am:satisfies` relation. They also both run a simulation that produces an output, `results.csv`. In the case of `a.emx`, the results show that the robot is able to move but does not detect the lines, thus the engineers add an `oslc_am:verifies` relation between the result and R1 and an `into:doesNoVerify` relation between the result and R2 (relations shortened to 'verif' and 'not verif' in the figure). The opposite result is true of the `b.emx` model, where it detects lines but can not move.

The final step in the example occurs when the project team decides to combine the best parts of the two models to produce a third. Thus in Figure 36, we can see that

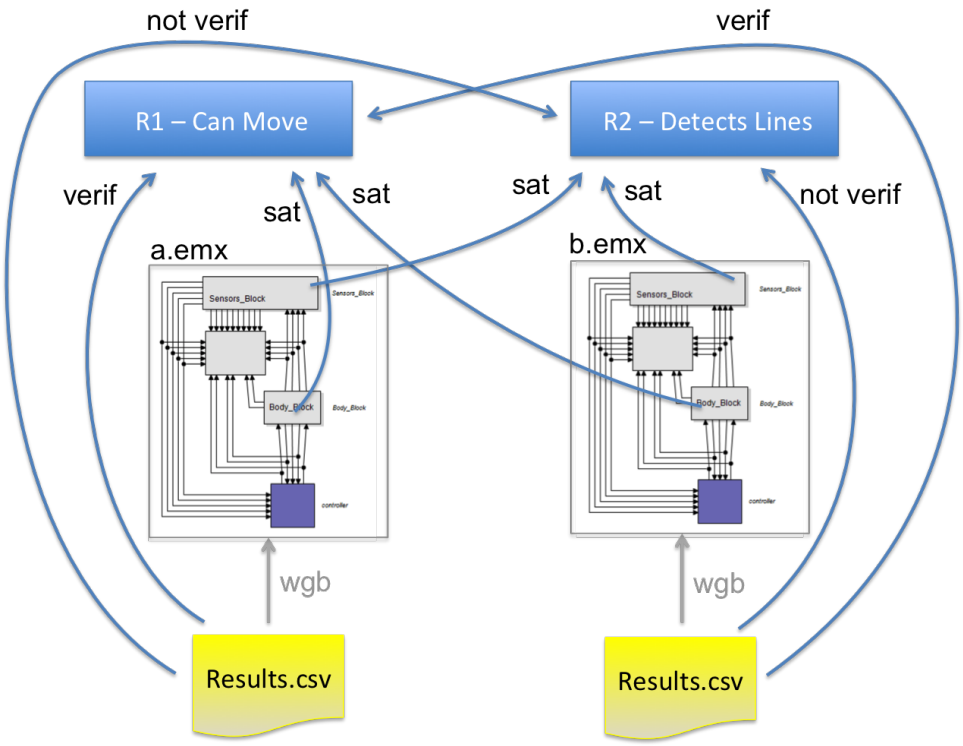


Figure 35: Two competing models and simulation results of the robot with links to the requirements.

a new model, `c.emx`, is created using from parts of `a.emx` and `b.emx`. The important relationship here and the use of the PROV wasDerviedFrom link between the submmodes in `c.emx` and the source models. In both of these relations the subtype `prov:quote` is used to show that it is a copy of the orginal submodel, thus we are able to trace where the submodels came from. Finally this model is simulated and produces a new result showing that it can both move and detect lines, the result is linked to both requirements via `oslc_am:verifies` relations.

The result of this traceability data being added is that the requirements engineer would be able to make queries that return either any models that implement specific requirement behaviour, or perhaps more importantly, find simulation results that provide evidence for or against the modelled system meeting those requirements.

The PROV concepts required to represent submodels have not yet been introduced. The starting point for this is a speciality of the standard PROV entity, a prov:Collection. The collection has members which other entities that it is considered to contain, thus we may define the collection entity that represents, as in Figure 37, a model file, `a.emx` that contains two submodels `body_a` and `sensors_a`. A key feature of this structure above naming submodels as properties of a plain entity, is that we may now represent the relationships between submodels in different files as first class constructs. In the example we see that there is also a modelling activity where the submodels of `a.emx` and `b.emx` and combined to form `c.emx`. We can now explicitly describe the provenance of the submodels in c by using the `wasDerivedFrom` relation, and thus show that the body submodel in c came from a and the sensors in c came from b.

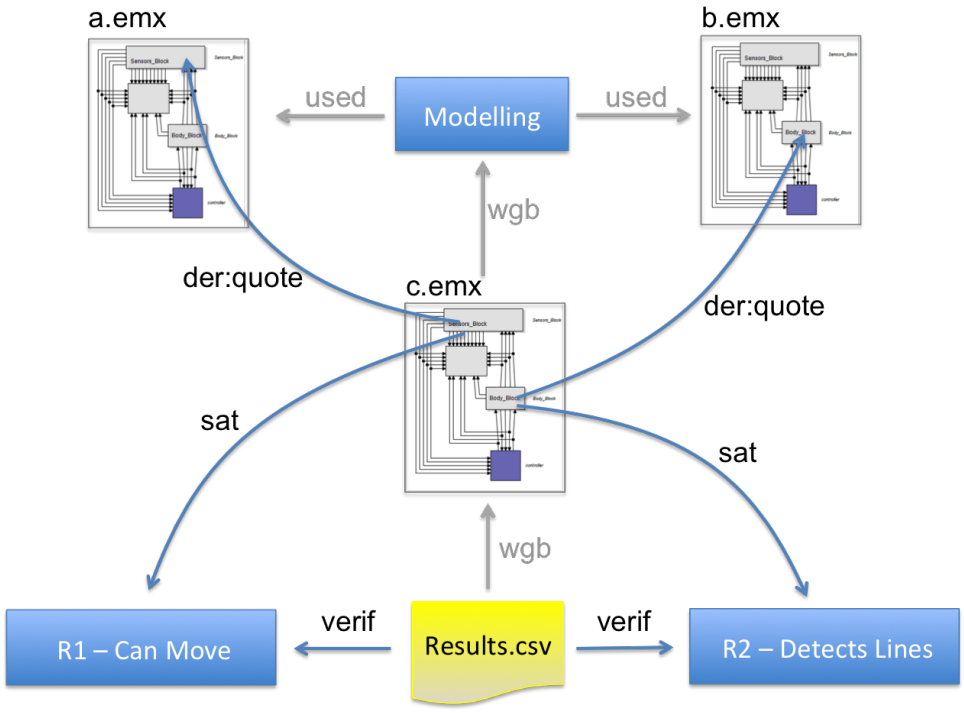
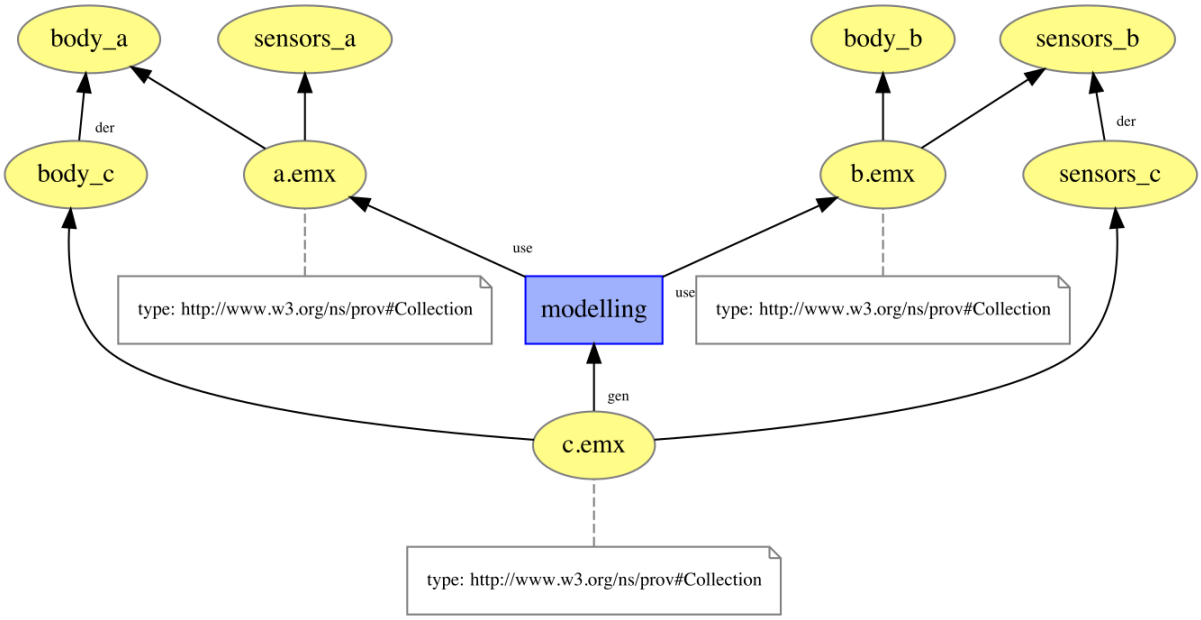


Figure 36: A final model of the robot that meets both requirements



(a)

```
entity(a.emx, [ prov:type='prov:Collection' ])
entity(sensors_a)
entity(body_a)
hadMember(a.emx, sensors_a)
hadMember(a.emx, body_a)

entity(b.emx, [ prov:type='prov:Collection' ])
entity(sensors_b)
entity(body_b)
hadMember(b.emx, sensors_b)
hadMember(b.emx, body_b)

activity(modelling)
used(modelling, a.emx,-)
used(modelling, b.emx,-)
wasGeneratedBy(c.emx, modelling, -)

entity(c.emx, [ prov:type='prov:Collection' ])
entity(sensors_c)
entity(body_c)
hadMember(c.emx, sensors_c)
hadMember(c.emx, body_c)

wasDerivedFrom(sensors_c,sensors_b, [prov:type='prov:Quote'])
wasDerivedFrom(body_c,body_a, [prov:type='prov:Quote'])
```

(b)

Figure 37: Prov representing submodels and their derivation in both graphic (a) and Prov-N (b) form.

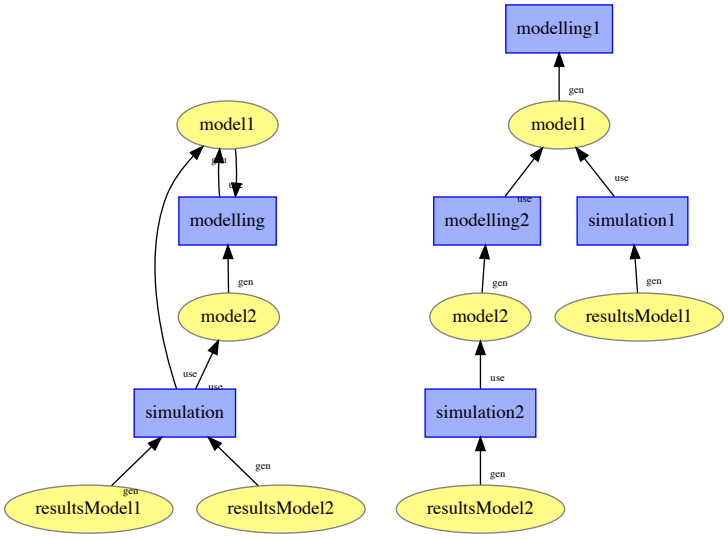


Figure 38: The same provenance with common activity names (a) and unique activity names (b)

A.6 UTRC Case Study Findings

As part of Task 3.5, Pilot Studies, UNew have been building an initial HVAC co-model based upon an Open Modelica model produced by UTRC. We have used this opportunity to manually record the provenance of the modelling process to see what insights this can give us that may inform how the provenance support in INTO-CPS should behave.

The first observation that was made very early on was that the entities and activities all must have unique identifiers and that these identifier can be rather long. In our original sketches of how the PROV might look, we had used activity names such as "simulation" and "modelling", but these can lead to ambiguities. In fact it is possible to use such short names in PROV-N and produce graphs, but doing this means that if we said modelling-used-file1, we are effectively saying that file1 was used in by an instance of the activity modelling, but we are not saying anything about that instance. Also, and more importantly, every time some modelling was performed it would be the same modelling activity that is connected to. The difference in clarity of the chronology of events can be seen by comparing Figure 38a, which uses common activity names and Figure 38b, which uses unique activity names to represent the same sequence of events.

Currently in the UTRC provenance the unique naming for entities has the form filename-subversion commit number, so for example one of the 20-sim fan coil unit models has the identity "fcu.emx-git-583" This has so far proven to be ok, but will fall down if a DSE run was performed resulting in multiple results files all with the same name but different paths, in which case the name will need to include the path as well. The first issue raised by this is that the type of file represented by an entity starts to become harder to discern for a user as we add more data into the ID, and slightly more trivially, the current tooling for visualising PROV-N uses the entity/activity's ID to name the element and so graphs could become much larger than needed. The suggestion here then is to use use the name-value properties afforded by PROV-N. The first use of this data would

be to hold a "type" value for each entity, such as "20-sim:modelfile" or "vdm:rt:class", this could make searching for types rather easier. A second use for the data would be to include a short name field, trivially this could be the file name. The same applies to the activities that appear in the provenance, these would also benefit from having types to indicate, for example, the nature of the simulation or modelling activity performed, e.g. "simulation:DSE" or "modelling:evolution", such types could be used in place of the short name for entities.

The final point that has become apparent is that attention will need to be paid to how the provenance handles model archives and duplication. The first step in the UTRC case study was to produce a standalone 20-sim version of the original Open Modelica Fan Coil Unit (FCU). Once produced, this model was copied into a Crescendo project folder and used to create a skeleton of a co-model. The skeleton Crescendo project was copied and had the DE controller and contract added, finally this Crescendo was exported from its archive for testing by another person. The key points are that, while it exists in multiple locations, the 20-sim FCU model was unchanged and this needs to be made explicit, also the provenance tooling and structure needs to allow for models to come out of archive for use and possible modification and also support these models being returned to that archive. For example, if the model exists in an archived form, is extracted and used to produce some simulation results and is then discarded, what should be recorded? It is possible to append PROV relations with name-value properties, so the solution currently employed in the UTRC provenance is to link the original archived file to the new copy via a *wasDerivedFrom* relation and applying a name value pair indicating that the derivation type was "copied". The use of archives is currently addressed by including the zip and any folders it contains as part of the path to the file.

A.7 Open Questions

There are still some large open questions that have not been addressed by the work so far.

How and when to capture data Having experienced manually creating provenance data there is certainly a need for tool supported data capture, but this does not address the issue of when does the engineer invoke the tool;

How should the provenance tooling behave should the tooling attempt to gather all models the engineer has been working on since it was last invoked or does it simply provide an interface to allow selection, or perhaps something in between possibly using known patterns of files to guide the engineer;

How and where should the provenance be stored The PROV fragments presented here are all encoded in the PROV-N text based notation, which could be used, or perhaps utilising either a relational or graph database such as Neo4J[45] would provide a better solution if we can successfully work with them in a distributed environment

What views would benefit the different stakeholders Most importantly, what views of the data are of value to support.

[45] `http://neo4j.com`