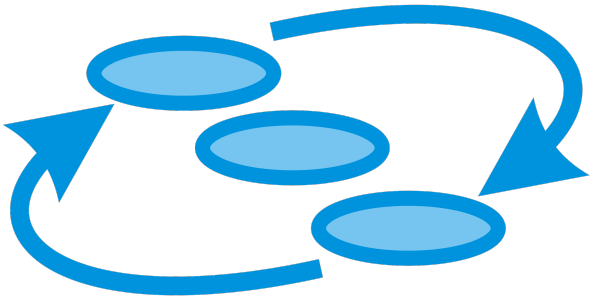
Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs



INTO-CPS

Methods Progress Report 2

Deliverable Number: D3.2b

Version: 1.0

Date: 2016

Public Document

Contributors:

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

Editors:

Carl Gamble, UNEW

Reviewers:

Christian König, TWT
Etienne Brosse, ST
Martin Peter Christiansen, 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	04-04-2016	Richard Payne	Initial document
0.2	1-11-2016	Ken Pierce	Workflows added
0.3	1-11-2016	Carl Gamble	DSE, Traceability added
1.0	15-12-2016	Carl Gamble & Richard Payne	Review comments addressed, appendices A & B added

Abstract

This document reports progress in Work package 3 (Multi-modelling Methods) in the second year of INTO-CPS, covering workflows, design space exploration, traceability, guidelines, and pilot studies. It contains four appendices, the first two outline specifications for the INTO-CPS SysML profile while the second two present updated versions of the INTO-CPS traceability ontology and an example of using the ontology concepts based upon the line follower robot pilot study.

Contents

1 Introduction

This report describes the work undertaken in Year 2 of INTO-CPS in Work Package 3. New and prospective users of the INTO-CPS technologies should refer to the Deliverable 3.2a Method Guidance 2 [FGPP16].

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.

This document outlines the progress made in the past year in these areas with further details of the current state presented in the appendices: Appendix A presents extensions to the INTO-CPS SysML metamodel; Appendix B presents the INTO-CPS DSE SysML metamodel; Appendix C presents the updated traceability ontology; and Appendix D presents an example model development for the purposes of identifying traceability and provenance data. These appendices are presented in this document, rather than Deliverable 3.2a [FGPP16] as they are either background material related to guidelines or snapshots of living documents – the material present does not constitute user guidelines.

2 Progress on WP3 Tasks

2.1 T3.1: Workflows

We had aimed to test full end-to-end use of the INTO-CPS in Year 2, however we overestimated when the integrated tools would be stable enough to handle such testing, particularly given the unforeseen need for WP4/5 to develop the INTO-CPS Application from scratch. The tool chain is now sufficiently mature that we can test end-to-end use early in Year 3 and generate guidance from our experiences.

Much of the effort in this regard was redirected into the "Tool Tsar" role established early in Year 2. The Tool Tsar role was created to ensure that the tools are able to support workflows described by WP3, and that releases to WP1 meet a minimum set of stability

and functionality requirements. To this end, we worked with WP4/5 to define a release procedure that gave time for testing. As part of the release procedure, we defined basic workflows that releases should support. These now form the "getting started" section described in D3.2a [FGPP16], along with the associated training materials. We regularly tested release candidates (RCs) against the pilot studies, and helped to ensure WP1 were warned of any updates that broke backwards compatibility, including providing examples of what needed changing in such instances. Next steps here are to ensure that this continues into Year 3 with both the tools and industrial case studies become more complex.

Regarding specific workflows, UTRC asked for guidance on how to model networks of distributed controllers in VDM. There is a well-established workflow in the VDM world of building a sequential modelling, then introducing concurrency and communication, and finally distribution across controllers in VDM-RT [LFW09]. There is however no current guidance on how to move to a multi-modelling context where distributed controllers described in a single VDM model become separate FMUs in a multi-model.

A first step here was to investigate how to model networked controllers in multi-models, which lead to the material on modelling networks in D3.2a [FGPP16]. This also led to specification being provided to WP4 on necessary extensions to the Overture FMU exporter to support this work. The next stage is to generate guidance on bridging the gap between this "DE-first" approach and networked FMU representation. We also intend to explore equivalents for CT-first and other workflows described in the Crescendo [FLV14].

2.2 T3.2: Design Space Exploration

In Year 2 of INTO-CPS it was identified that the definitions guiding the DSE scripts just appear with no meaningful links to the any other artefacts in the tool chain. This has two effects, firstly there is no traceability back to the requirements from which we might understand why the various objectives (measures) were being evaluated or why they were included in the ranking definition. Secondly, since the configurations were created manually for each new DSE experiment it is easy to imagine that the DSE analysis and ranking might not be consistent among the experiments. To address this issue it was decided to make the definition of the objectives of DSE explicit early on, i.e. in the architectural phases, of the INTO-CPS workflow by creating a DSE SysML profile [FGPP16]. The profile (designed using a metamodel in Appendix B contains a number of views allowing the definition of the objectives to be computed, their links back to requirements, the method for ranking and the connections between the multi-model and the analysis such that the required data may be obtained. The purpose of the view is to support traceability back to requirements, facilitate description of the DSE analysis to stakeholders and to permit automated generation of consistent DSE configurations.

The second year has also seen the development of the first closed loop DSE script in INTO-CPS. The scripts themselves are reported under T5.1 and described in [Gam16], however the genetic script has options both in terms of how the algorithm works and also parameter that may be tunes for different design spaces. Thus experiments varying the algorithm options and parameters have been run with a set of design spaces based upon the line follower robot and three tank water tank models. These experimental results,

which at the time of writing are being compiled measure the costs of DSE in terms of the number of simulations performed to obtain a result and the quality of that result by comparing the Pareto front obtained with one obtained from an exhaustive search. These results can be used for the basis of guidance by users regarding which algorithm to use, what parameters and how many times to run a script to obtain the best results.

2.3 T3.3: Provenance and Traceability

The second year of traceability work has seen the abstract ontology work in year one progress towards implementable design through the development of concrete examples. The line follower robot has once again proved its worth by acting as the subject of this example which runs from the elicitation of requirements through to simulation results and a design change based upon those results. The example, which may be found in Appendix D describes the individual steps taking the design from requirements through to results focussing on the activities performed, the relations that need to be captured and giving concrete examples of the messages in the JSON format.

The ontology that describes the relations the INTO-CPS aims to capture has also been updated to maintain step with the rapidly changing tool chain.

Validation of the task efforts is important here to ensure they are in line with the industrial partner needs. To this work package 1 have taken part in both written surveys and also more recently structured interviews conducted over the telephone. The results of the structured interviews, which are currently being reviewed, have revealed a strong desire for a prescriptive approach to traceability where the engineer is directed toward the important relations needed to support their traceability goals.

2.4 T3.4: Guidelines

In the second year of INTO-CPS, we had several objectives. First we maintained the concept base created in the first year – including new terms as required based upon new developments in the project, and ensuring any disagreements with existing terms are rectified. The concept base has also yielded a common glossary to be used in the project, separate to Deliverable 3.2a [FGPP16] containing the concept base.

This concept base has been circulated to the project – first to Work Package leaders, and subsequently to the remainder of the project, with discussion following in relation to Hardware-in-the-Loop and Software-in-the-Loop testing. These caused difficulties due to the terms being used in both industry and test automation, with subtly different views. As the concept base is to be used for the project, it was decided that we use the term as associated in the Test Automation task (T5.2). Any alternative use of the term should therefore acknowledge that it differs from the project view.

The second aim is to ensure guidelines on various project technologies are made available and accessible within the project – especially for industrial users. Therefore the task managed a series of webinars, hosted by TWT. To date, there have been four webinars:

concepts; joint WP3 and WP4 tutorial on co-simulation[1]; workflows; and design space exploration. A webinar for traceability was planned, but has been moved to next year to ensure that usable technology is available.

Finally, the task worked with Task 3.1 on guidelines for requirements engineering and with T2.1 on extensions to the INTO-SysML profile (reported in Appendix A and Deliverables D3.2a [FGPP16] and D4.2c [BQ16]). The requirements engineering guidelines use outputs from the traceability and provenance task (T3.3) and from responses from the T2.1 industrial partner survey from year 1 of the project.

2.5 T3.5: Pilot Case Studies

In Deliverable D3.4 [FGP+15], produced in the first year of the project, we proposed a roadmap to expand coverage of the pilot study compendium. Deliverable D3.5 [PGP+16] presents an updated collection of pilot studies that aim to meet the identified shortcomings and proposals of newly available approaches from the technology providers in INTO-CPS. In addition, a large effort of this task was to migrate pilots from baseline technologies to the INTO-CPS technologies as they were made available.

The migration aspect took some effort, as technologies were not always stable, and bugs were identified and rectified. As such, this took more time than was previously assumed.

The main shortcoming identified was the lack of network communication in the pilot studies. As such, along with T3.1 we have developed the *Ether* pilot which may be used to model network-based communication. We have begun including this in the *UAV Swarm* pilot, however, issues with the COE have caused difficulties. This aspect of the study is therefore currently withheld from the publicly accessible examples.

The pilots provide coverage of all INTO-CPS simulation technologies (VDM-RT, 20-sim and OpenModelica), have architectural models in SysML using the INTO-SysML profile, may be co-simulated with the INTO-CPS application, can perform DSE and have support for test automation. We currently do not support code generation or traceability in the pilot studies. Preliminary examples exist within T3.3 and T5.4 respectively, however the tool support is not mature enough; therefore they have not been integrated into the pilots compendium.

As stated in Deliverable D3.5 [PGP+16], the focus for the final year is to propagate the network communication in existing studies (UAV Swarm and Smart Grid are natural candidates), to support additional features of the INTO-CPS tool chain as they become available and to add to the compendium with new studies as appropriate.

[1]This was produced prior to the development of the INTO-CPS Application, and described the tasks for architectural modelling, simulation modelling and co-simulation

References

[AdLG15] Nuno Amálio, Juan de Lara, and Esther Guerra. FRAGMENTA: A theory of fragmentation for MDE. In *MODELS 2015*. IEEE, 2015.

[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.

[BQ16] Etienne Brosse and Imran Quadri. SysML and FMI in INTO-CPS. Technical report, INTO-CPS Deliverable, D4.2c, December 2016.

[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.

[FGPP16] John Fitzgerald, Carl Gamble, Richard Payne, and Ken Pierce. Method Guidelines 2. Technical report, INTO-CPS Deliverable, D3.2a, December 2016.

[FLV14] John Fitzgerald, Peter Gorm Larsen, and Marcel Verhoef, editors. *Collaborative Design for Embedded Systems – Co-modelling and Co-simulation*. Springer, 2014.

[Gam16] Carl Gamble. DSE in the INTO-CPS Platform. Technical report, INTO-CPS Deliverable, D5.2d, December 2016.

[LFW09] Peter Gorm Larsen, John Fitzgerald, and Sune Wolff. Methods for the Development of Distributed Real-Time Embedded Systems using VDM. *Intl. Journal of Software and Informatics*, 3(2-3), October 2009.

[PGP+16] Richard Payne, Carl Gamble, Ken Pierce, John Fitzgerald, Simon Foster, Casper Thule, and Rene Nilsson. Examples Compendium 2. Technical report, INTO-CPS Deliverable, D3.5, December 2016.

A INTO-CPS SysML Metamodel Extensions

We have extended the INTO-CPS SysML metamodel and profile to allow a multi-model to include an FMU for visualisation and the environment, the addition of the CComponents type and an extension for FlowPorts to support minimum and maximum values in later versions of the SysML-FMI translation.

In this appendix we present the extensions to INTO-SysML in metamodel fragments as presented in [APCB15]. In Figure 1, we extend the F_Blocks metamodel fragment to include CComponents, additional ComponentKind enumerations and the repositioning of Flowport ownership – changes are shown in red. The diagram shows that a `CComponent` is a subtype of `Component`, and contains no additional attributes. To support modelling of non-CPS components, three extra `ComponentKind` enumerations are added: `environment`, `visualisation` and `composition`. Finally, where previously the `Block` stereotype was the owner of `FlowPorts`, we have adjusted that relationships so that neither `System` or `CComponent` instances can own `FlowPorts`.

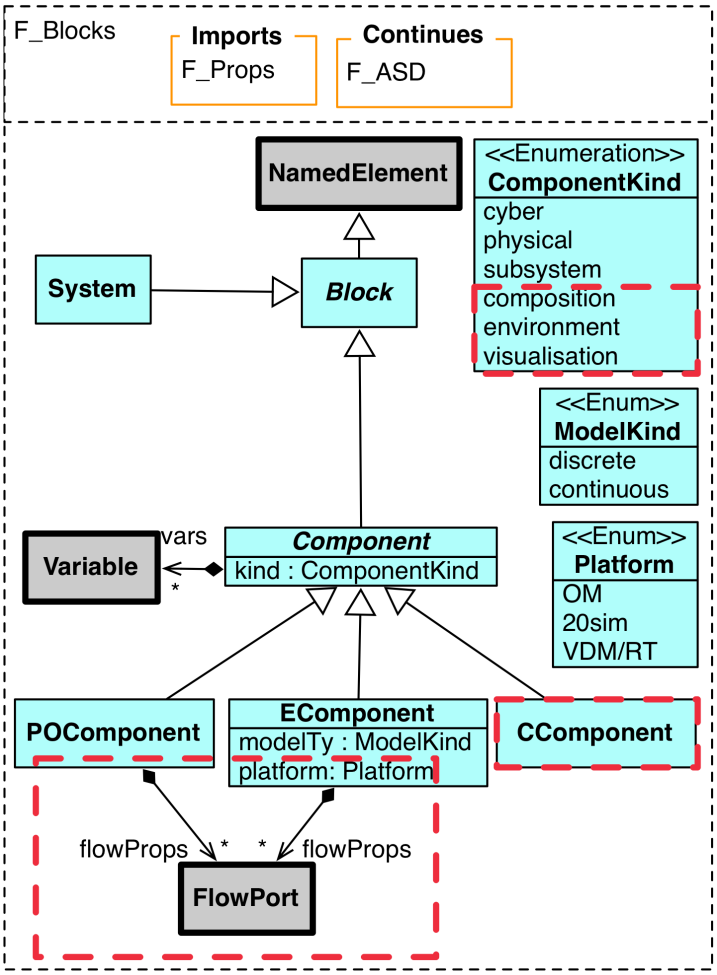


Figure 1: Extension to F_Blocks fragment – extensions are highlighted with a red dotted line

A minor update to the `FlowPort` definition is shown in Figure 2. The extension is to allow minimum and maximum values for flowports to be defined. The `min` and `max` attributes are of type PType – the primitive types of INTO-SysML.

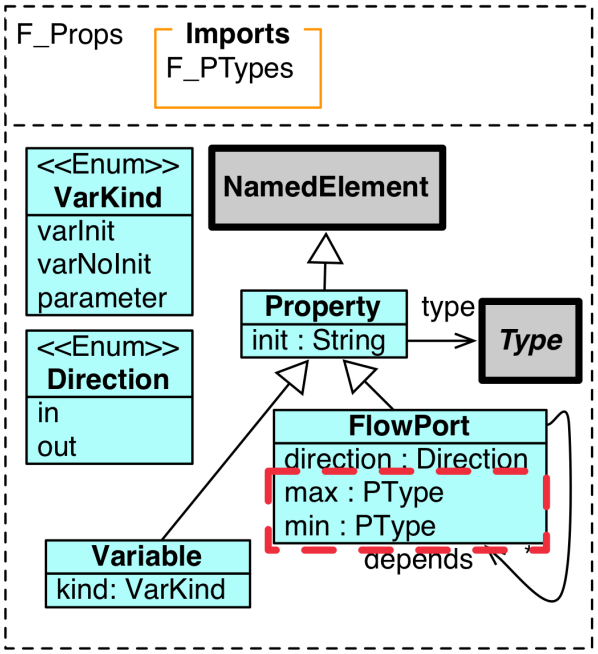


Figure 2: Extension to F_Props fragment – extensions are highlighted with a red dotted line

The metamodel extensions are realised in SysML profile diagrams in Deliverable D4.2c [BQ16].

B Design Space Exploration Metamodel

The metamodel for the DSE SysML extensions is presented in three fragments (we use Fragmenta [AdLG15] as in Deliverable D2.1a [APCB15]). The first fragment, in Figure 3, shows the elements required for defining DSE *Parameters*. A `DSEAnalysis` block is defined along with several `Parameters` and `ParameterConstraints`. Parameters have values and refers to Component Variables. ParameterConstraints have a defined predicate.

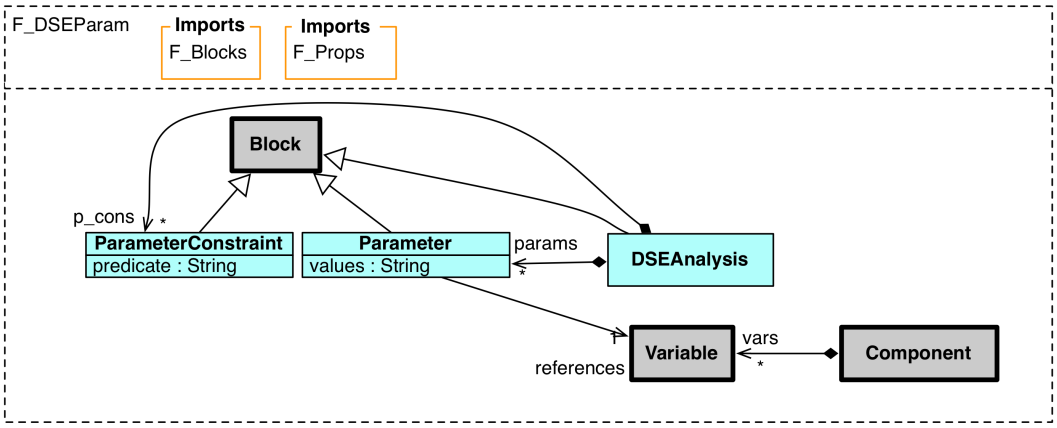


Figure 3: Metamodel for DSE Parameter Diagram elements

The DSE *Objective* elements are defined in Figure 4. A `DSEAnalysis` block is defined, as in the `DSEParameterDiagram` with a collection of `Objectives` and `ObjectiveConstraints`. Objectives have a collection of `FlowPorts` that are related to the `FlowPorts` owned by component `BlockInstances` of the INTO-SysML profile. The metamodel dictates two specialisations of Objectives – `ExternalScripts` which have an associated file name, and `InternalFunctions` which have a function type. ParameterConstraints have a defined predicate.

The final metamodel fragment, in Figure 5, shows the model elements of for DSE *Ranking*. This fragment also uses a `DSEAnalysis` block and a collection of `Ranking` blocks. At present only the `Pareto` ranking is supported, which contains several `ParetoValues` – each with a direction.

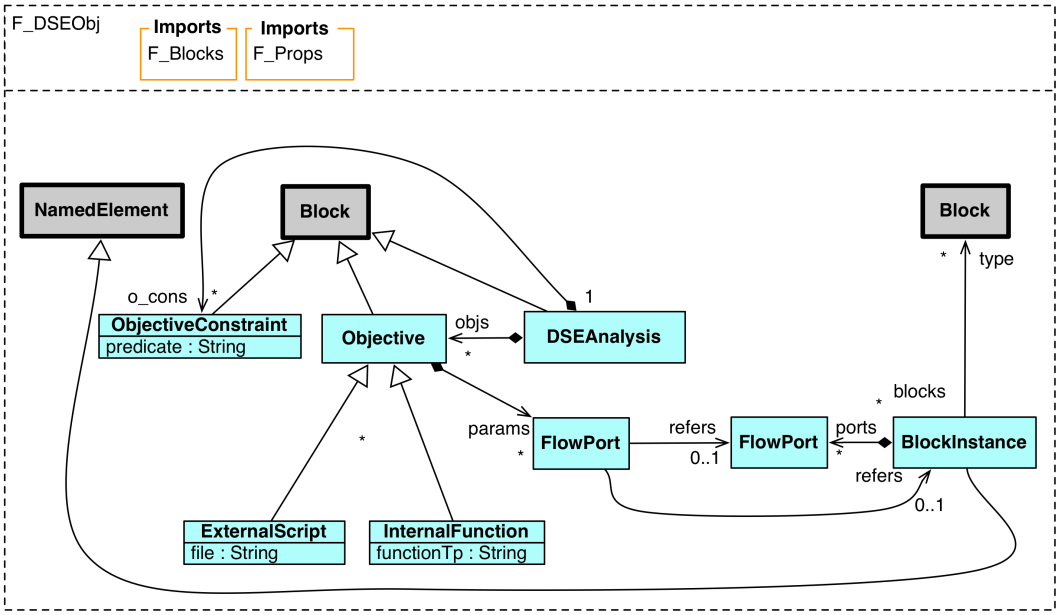


Figure 4: Metamodel for DSE Objective Diagram elements

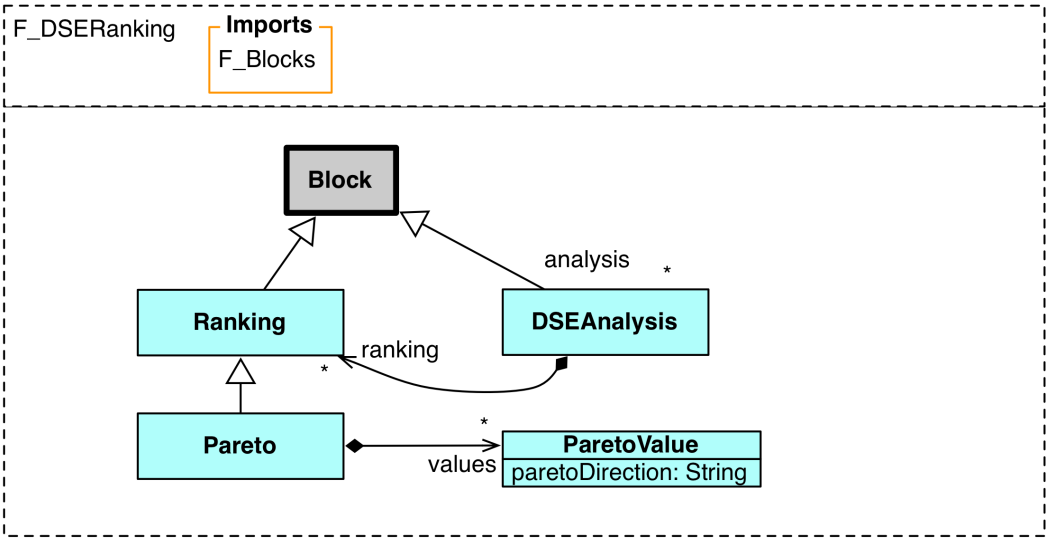


Figure 5: Metamodel for DSE Ranking Diagram elements

C Updated INTO-CPS Traceability Ontology

In this section we 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. The figures contain a collection of SysML blocks: blue element represent an *activity*; a yellow element represents an *entity*; and an orange element represents an *agent*.

C.1 Requirements

Starting with requirements, Figure 6 shows the activity of requirements engineering. This activity makes use of *design notes* and *requirement sources*, which are the primary documents from the stakeholders and it produces *requirements*. The *requirements* correspond to the requirements defined in the *Requirement Definition View* as described in Deliverable D3.2a [FGPP16]. Note that the requirement itself has two OSLC relations to itself, these are to support the relationships between individual requirements to be recorded, only two are shown here 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* – these documents may be SysML models or Excel files also described in Deliverable D3.2a [FGPP16].

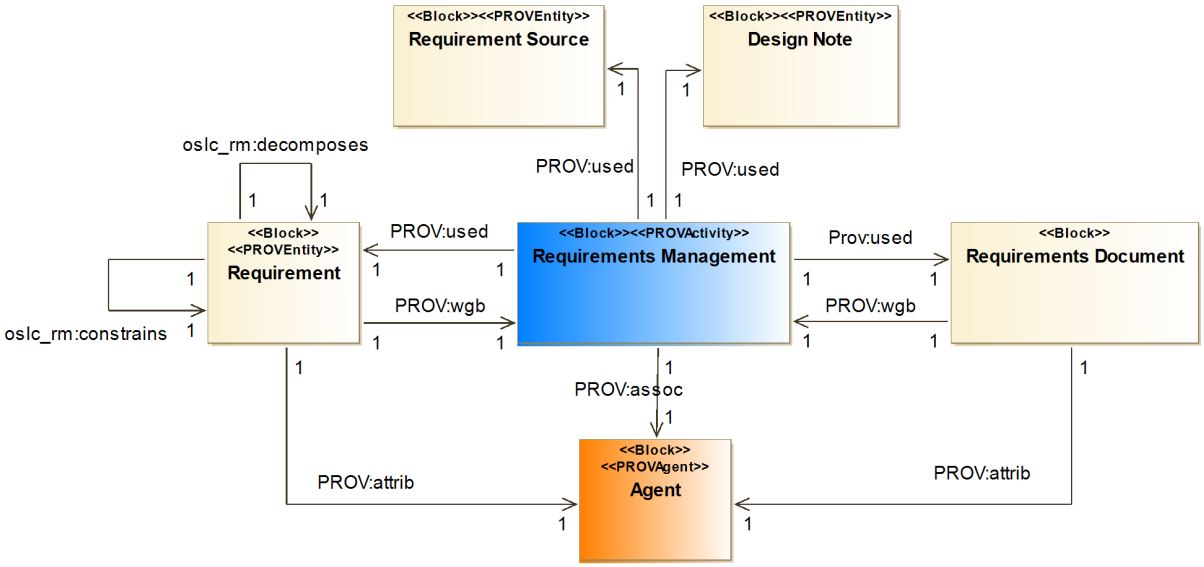


Figure 6: Block Definition Diagram (BDD) of the requirements activity

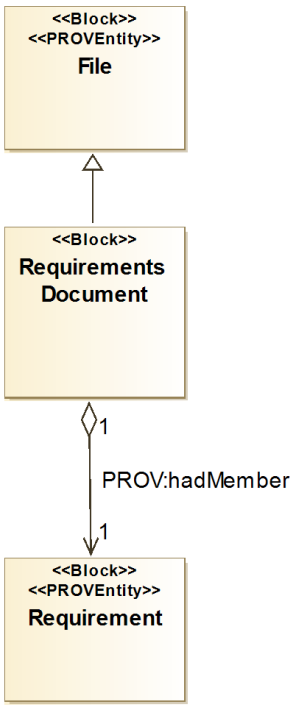


Figure 7: BDD of the requirements files

C.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 *connection diagram*) and the *architecture subsystems* they contain. The architecture subsystems may be related to any requirements they satisfy. The files that represent the architecture are shown in Figure 9.

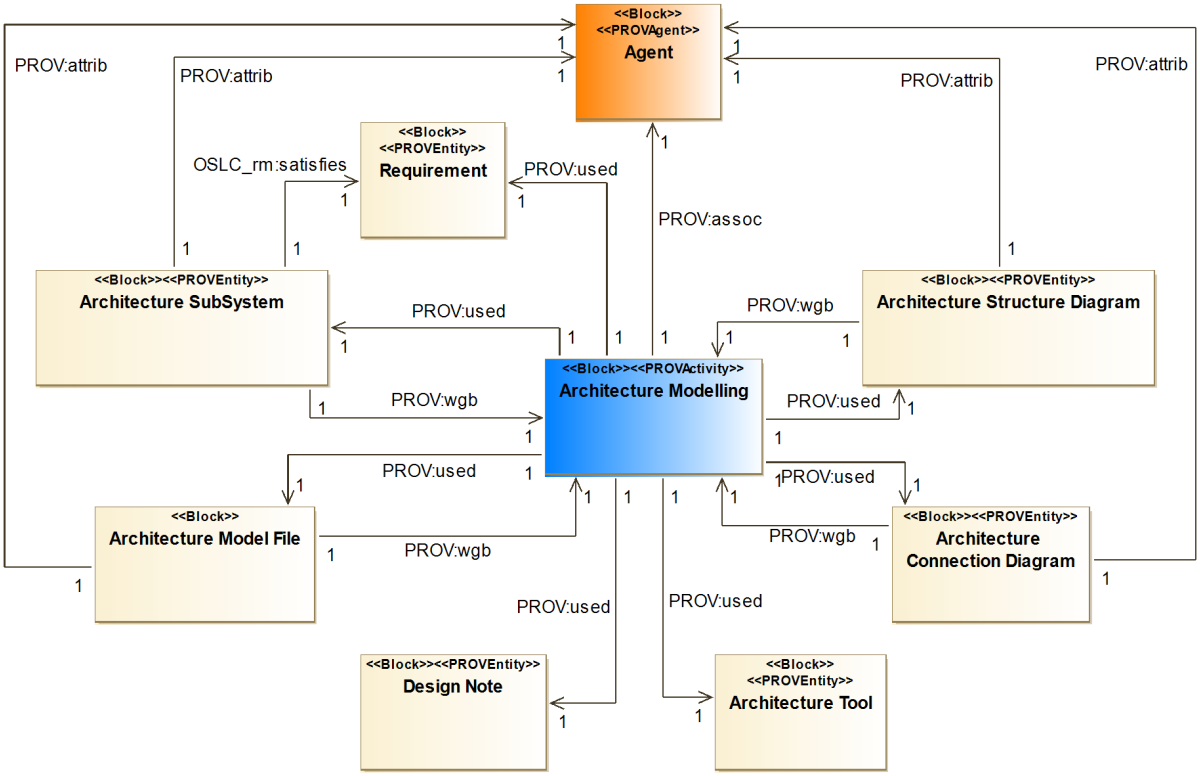


Figure 8: BDD Architecture modelling

The activity of *architecture configuration creation* is presented in Figure 10. Architecture configuration creation uses the architecture model to generate an *architecture configuration*.

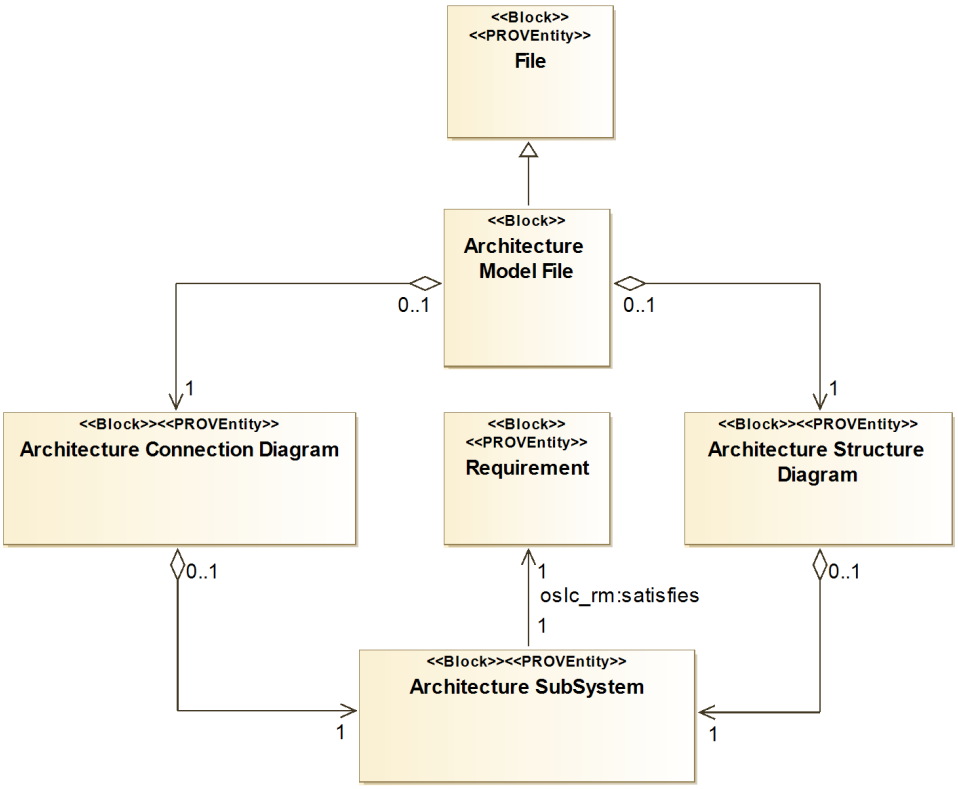


Figure 9: BDD Architecture modelling elements

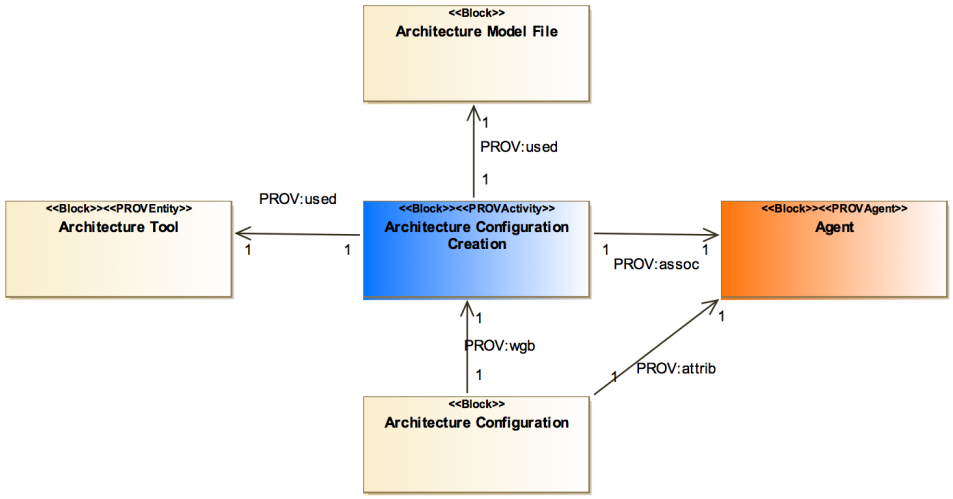


Figure 10: BDD Architecture modelling elements

C.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 11. This functionality is provided by the *INTO-CPS Application*.

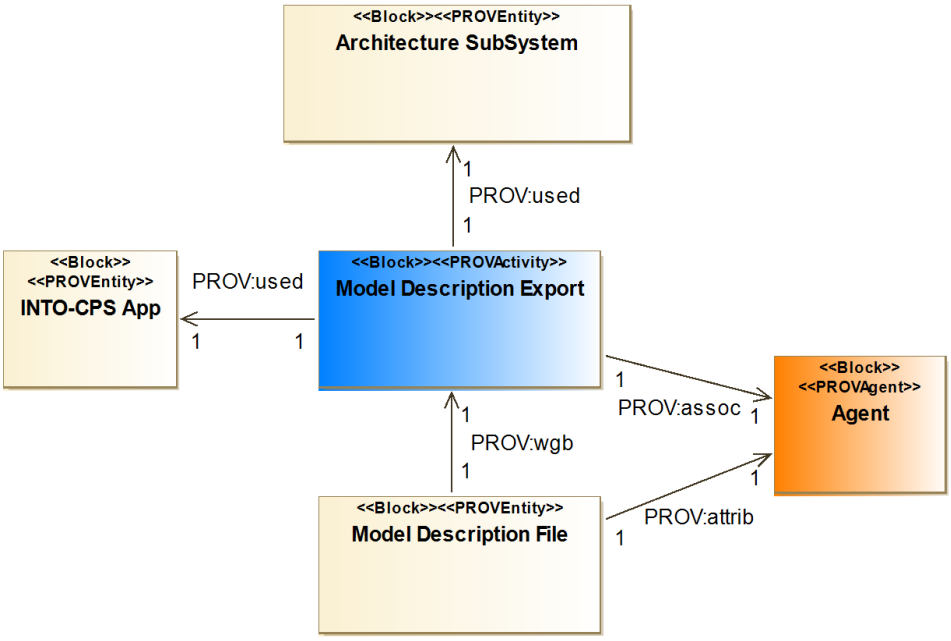


Figure 11: BDD of the model description export activity

C.4 Simulation Models

There are two distinct activities shown in Figure 12, *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 13 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.

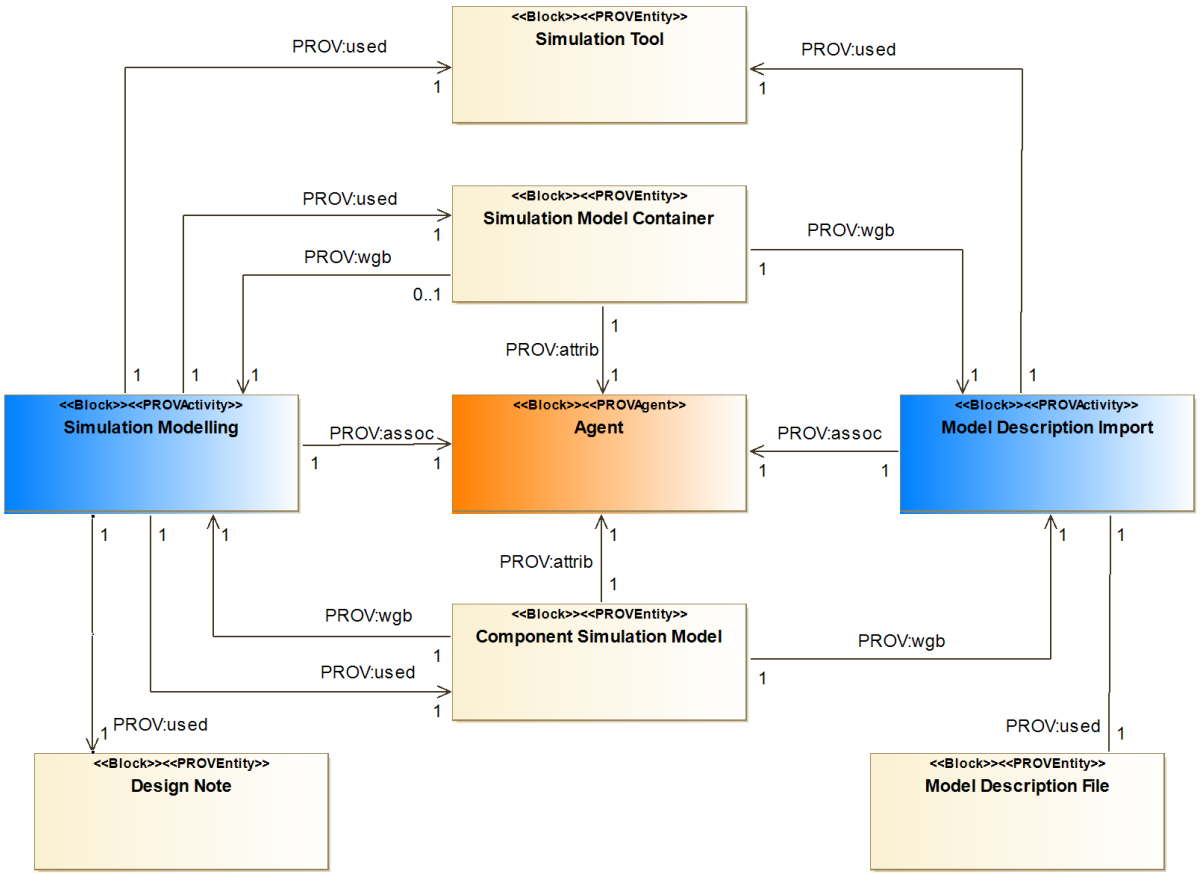


Figure 12: BDD showing simulation model creation activities

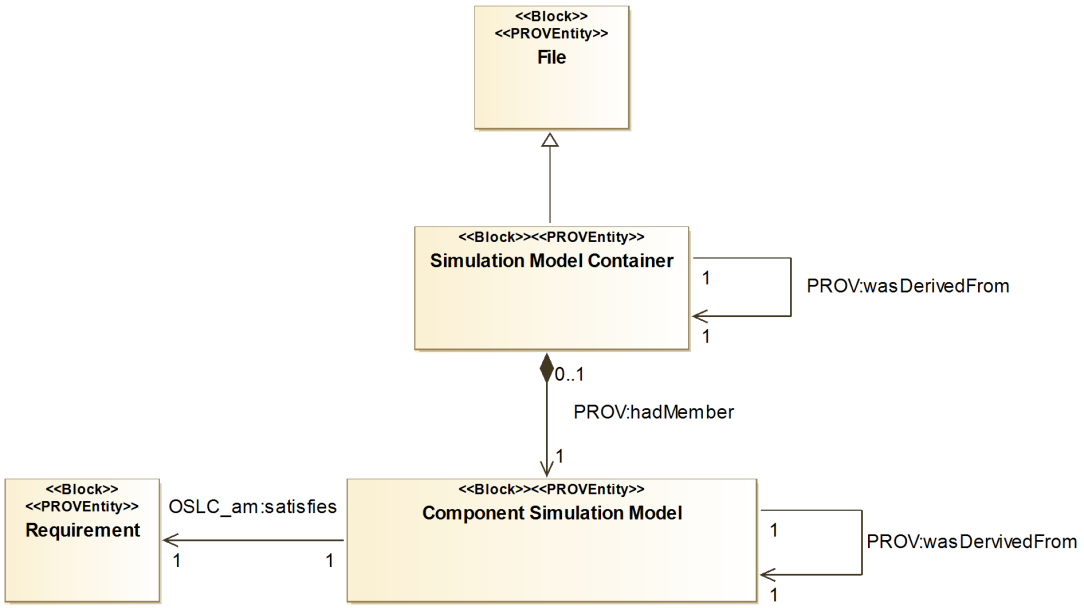


Figure 13: BDD showing simulation model files

C.5 Model Checking

Model checking (Figure 14)begins with an identical structure (*model description import* and *model check modelling*) as we saw for simulation models previously. Figure 15 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 16

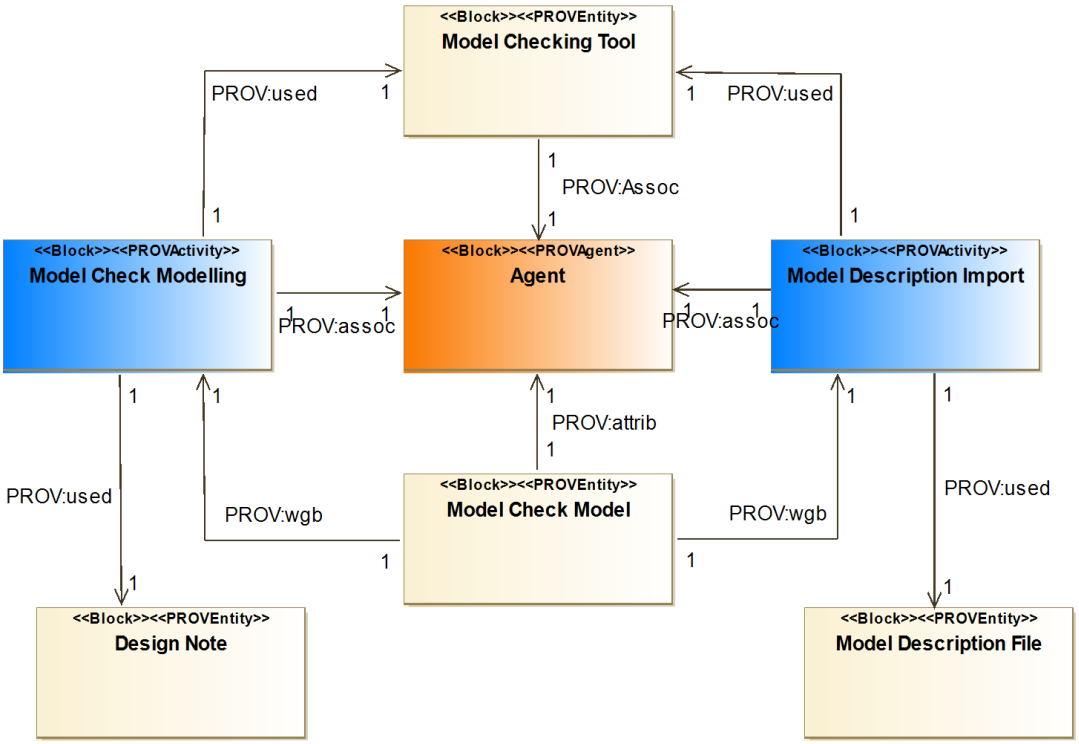


Figure 14: BDD showing model check model creation

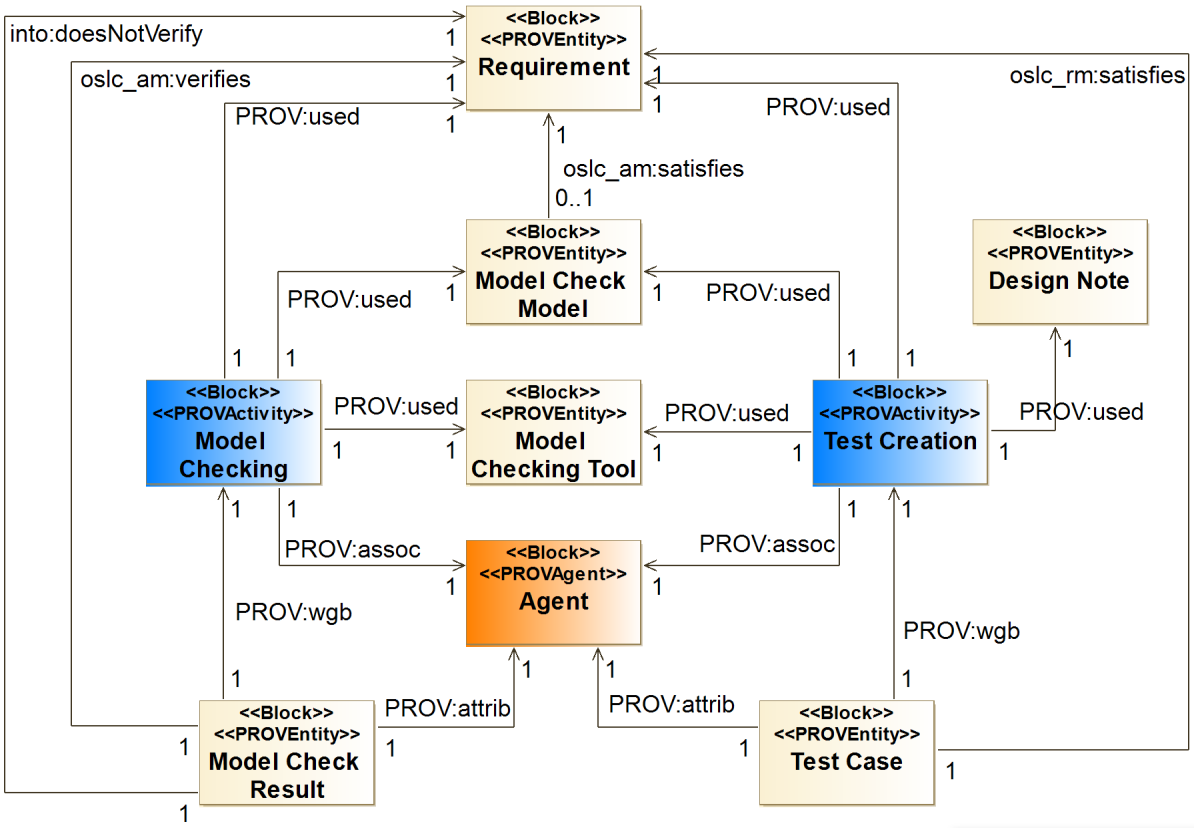


Figure 15: BDD showing the activities of model checking

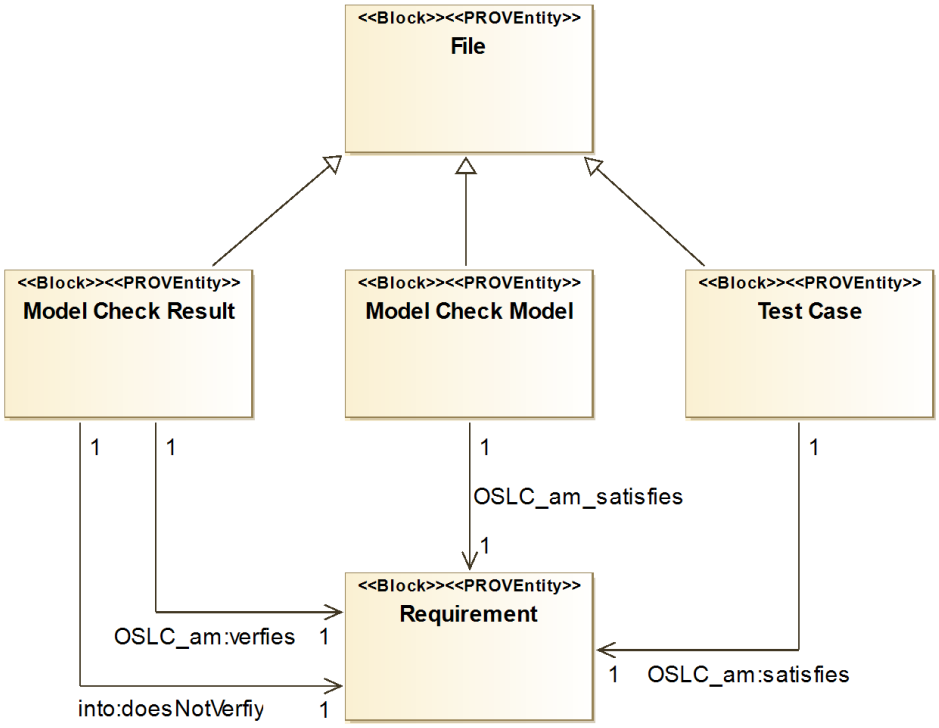


Figure 16: BDD showing the file elements around model checking

C.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 17 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.

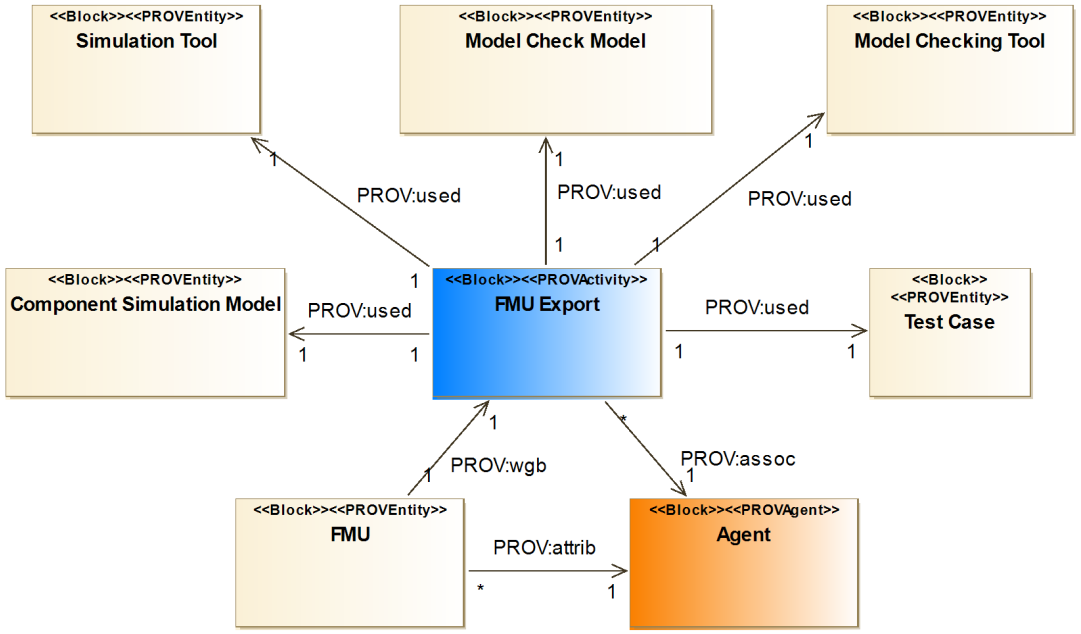


Figure 17: BDD showing simulation FMU generation

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 18. 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 19, 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 20.

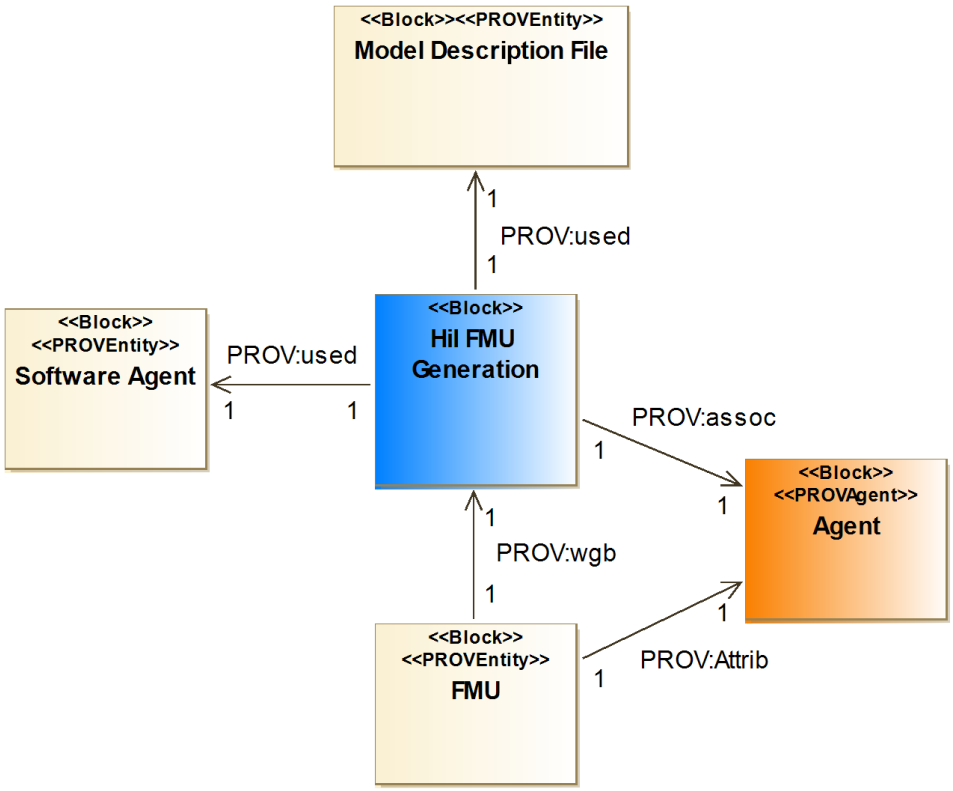


Figure 18: BDD showing FMU generation to allow HiL simulation

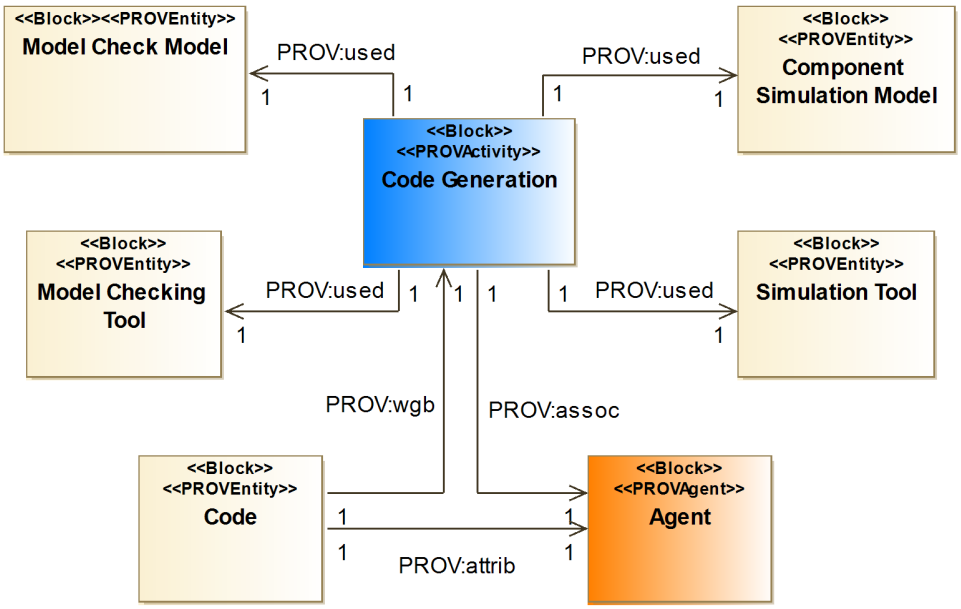


Figure 19: BDD Showing the activity of code generation

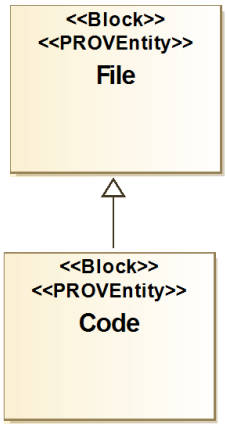


Figure 20: BDD showing the files associated with code generation

C.7 Simulation

Before simulation can take place we must produce a multi-model configuration file, this activity is shown in Figure 21. Here the INTO-CPS Application uses the generated FMUs and the architecture configuration to produce a *multi-model configuration*.

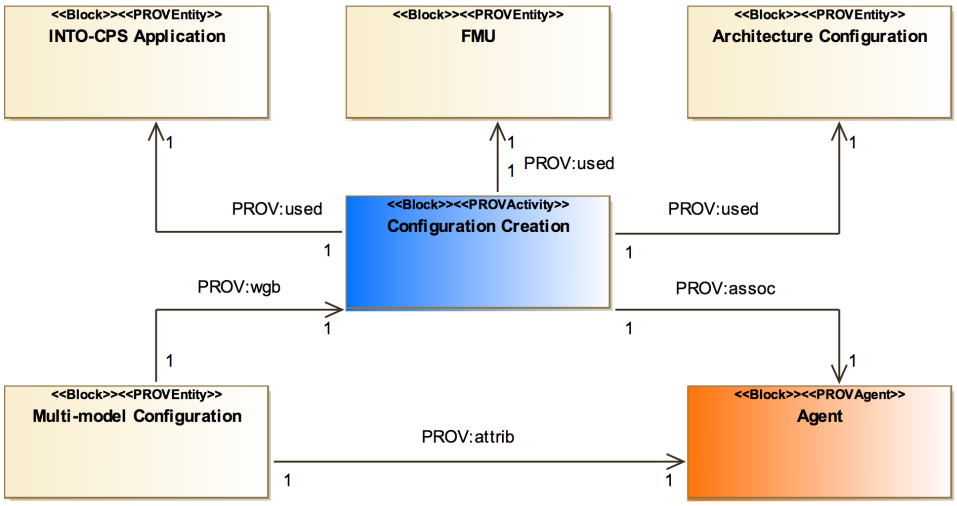


Figure 21: BDD showing the multi-model configuration activity

The *Simulation Configuration Creation* activity uses the aforementioned multi-model configuration to create a simulation configuration, shown in Figure 22.

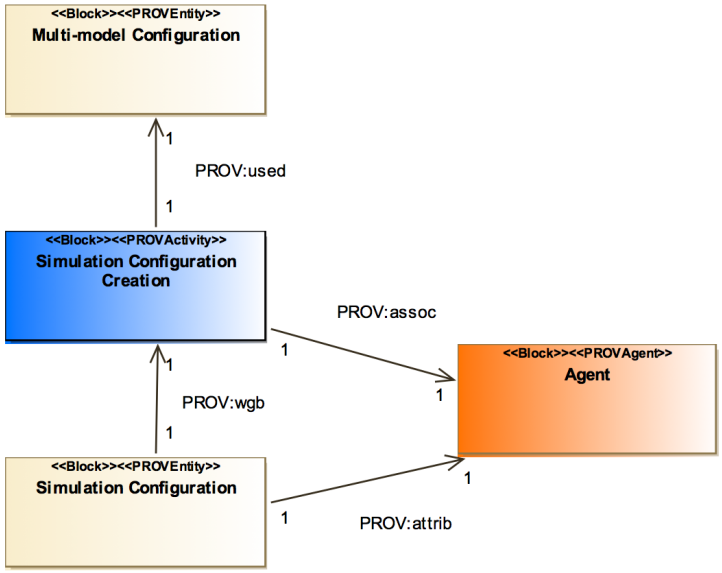


Figure 22: BDD showing the simulation configuration activity

Figure 23 shows the activity of *simulation* itself. Here we see that it makes use of the COE, multi-model configuration, simulation configuration, and the FMUs that have already been generated. Here we also see that the FMUs themselves may make use of component simulation models, and simulation tools if the FMU is a wrapper or may reference a hardware asset if it is a HiL FMU.

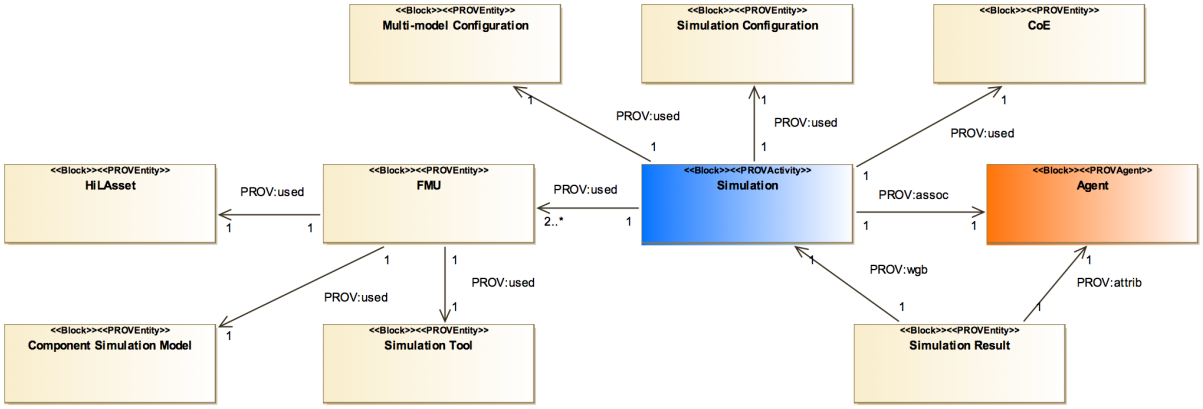


Figure 23: BDD showing the activity of simulation

Finally, Figure 24 shows the files associated with simulation. The simulation result may be linked to a requirement, providing evidence for or against that requirement being met.

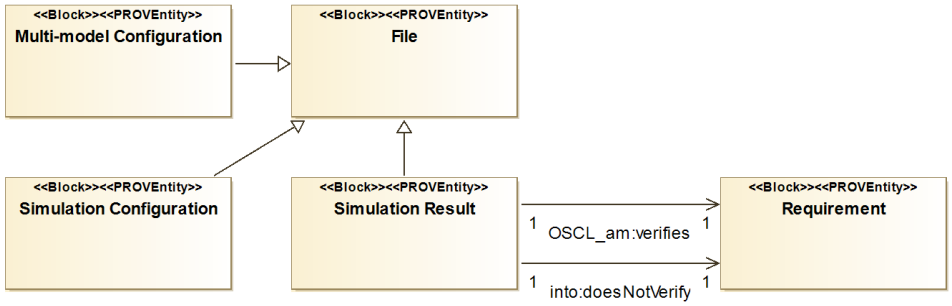


Figure 24: BDD showing the files associated with simulation

C.8 DSE

Figure 25 shows the *DSE configuration creation* activity. The activity takes a multi-model configuration and queries the user to define the range of parameters and algorithm choices for the actual DSE itself (output in the DSE search and DSE analysis configurations).

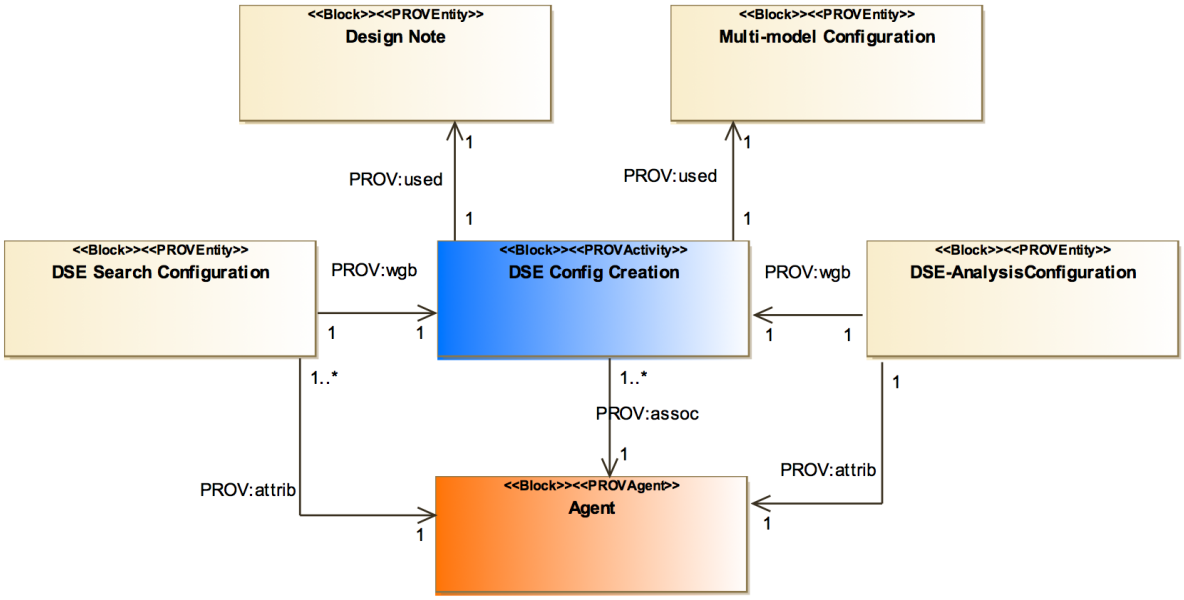


Figure 25: BDD showing the DSE configuration

Figure 26 shows the *DSE Analysis Creation* activity, which generates a DSE analysis script and scenario data.[2]

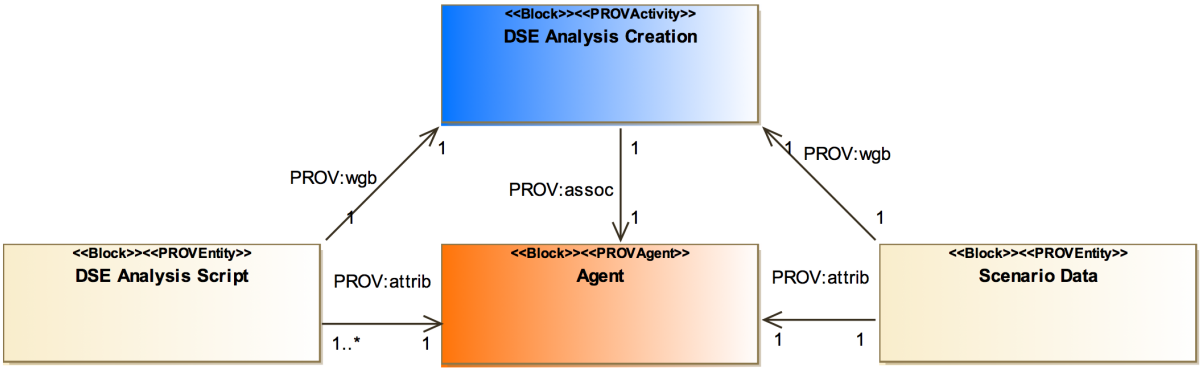


Figure 26: BDD showing the DSE configuration

The *DSE* activity (shown in Figure 27) uses the multi-model, simulation, DSE search and DSE analysis configurations to select the appropriate scripts and scenarios to launch simulations, evaluate the objective values of each simulation, and then rank them as a result.

[2]Both this activity and the *DSE configuration creation* activity will be updated in Year 3 to reflect the use of the INTO-CPS DSE SysML profile as described in Deliverable D3.2a [FGPP16].

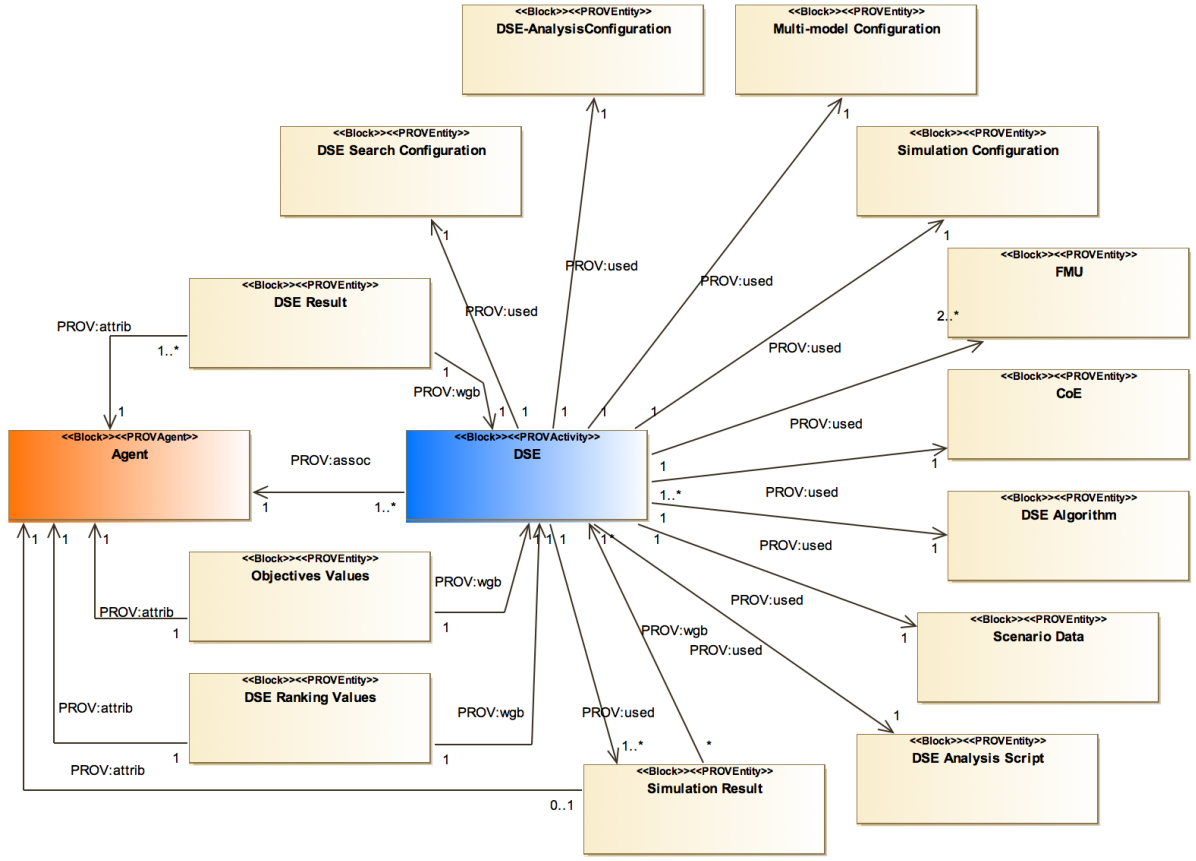


Figure 27: BDD showing the DSE activity

Finally in DSE, Figure 28 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.

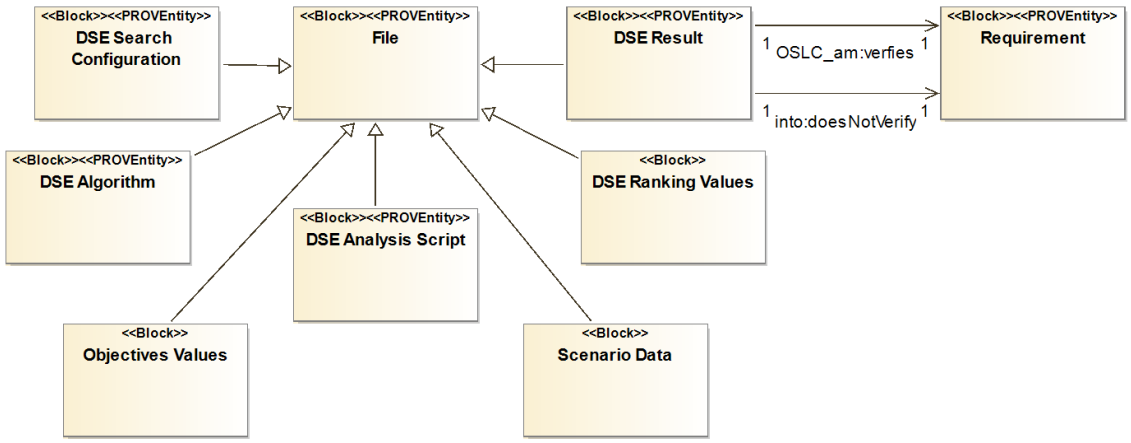


Figure 28: BDD showing the DSE files

C.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 29, 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 30 shows the files associated with design notes.

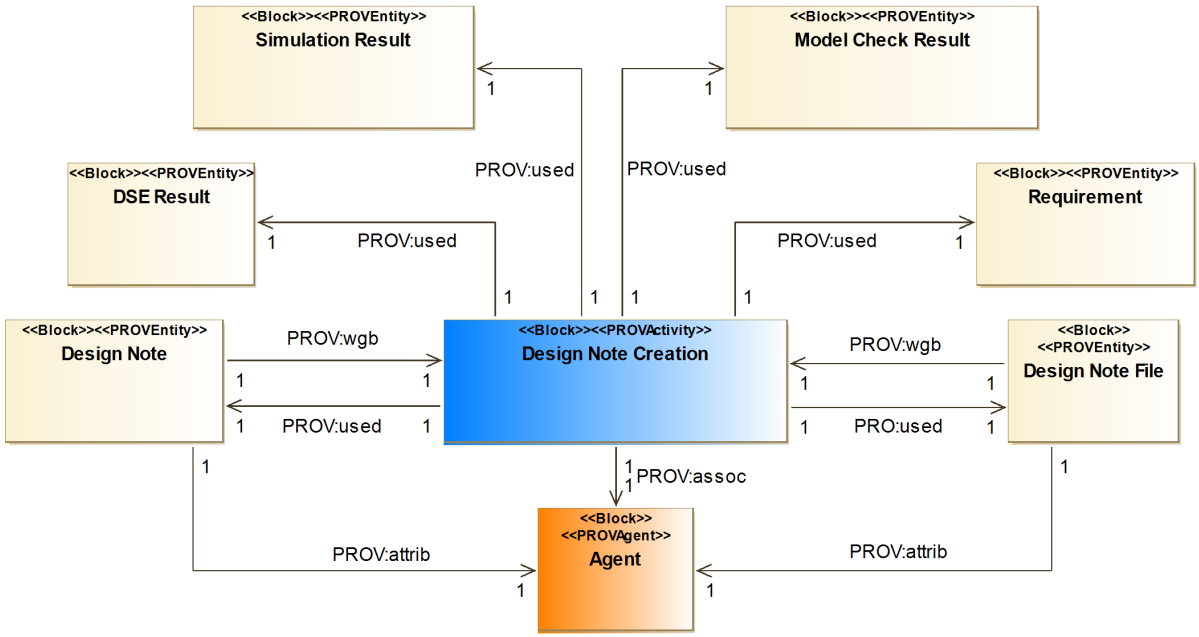


Figure 29: BDD showing design note creation

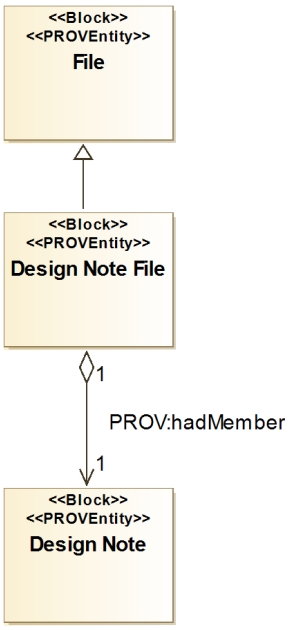


Figure 30: BDD showing the design note files

D Line Follower Example

D.1 Introduction

The robot example presented here is inspired by the work that took place to produce the line following robot pilot study, shown in Figure 31. The example depicts a fictitious version of the development of this robot model starting with the generation of requirements from a stakeholder document through to the simulation of a model that meets those requirements. The example depicts the development of models from scratch and also the reuse of a library model.

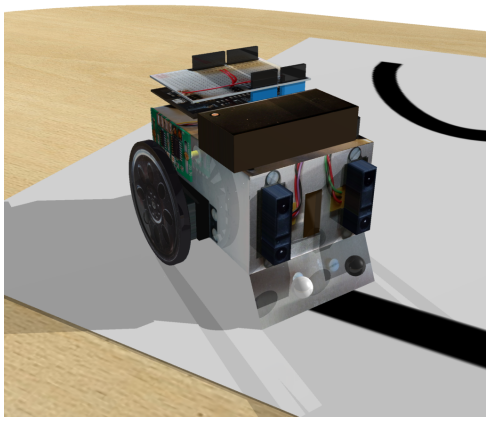


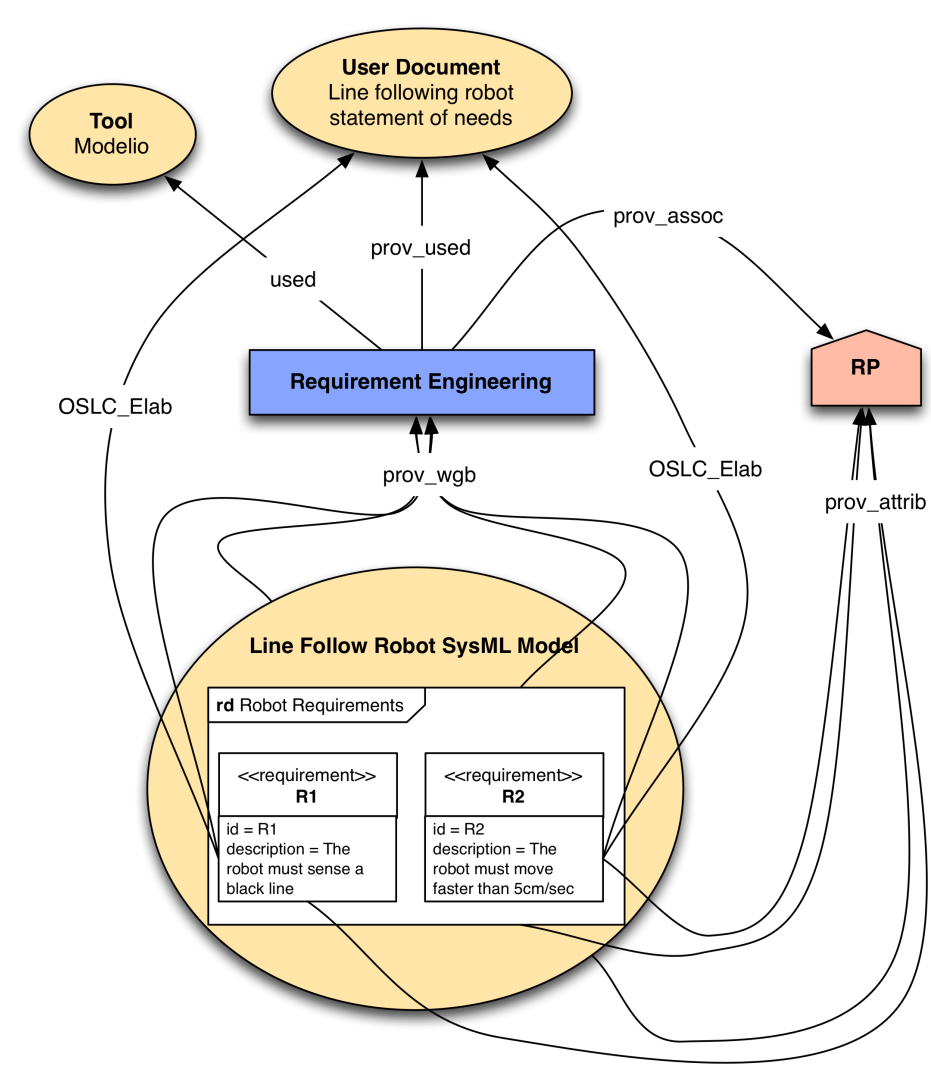
Figure 31: 3D view of the line follow robot generated by its co-model.

The example is broken down into steps at the granularity of the activities defined in the traceability ontology, Appendix C, and shows the modelling artefacts (entities) that

are used and generated while performing those activities. Each step includes a text description of the activities that are performed, the OSLC triples that would record the activity and a sequence of data gathering and archiving steps required.

D.2 User docs → Requirements

D.2.1 Element Graph



D.2.2 Textual Description

Line following robot statement of needs is the user document entities which inform a model requirements. In this step the *Requirements Engineering* activity used those *User Documents* and the *Modelio* tool. The *RobotSysML* model, *Requirements Diagram* and requirements *R1 (the robot must sense a black line)* and *R2 (the robot must move faster than 5cm/sec)* were generated by the *Requirements Engineering* activity which elaborate the needs identified in the *User Documents*. Finally, the *Requirements Engineering* activity is associated with the agent *RP* and the *RobotSysML* model, *Requirements Diagram*

and requirements *R1* and *R2* are attributed to the agent *RP*.

D.2.3 Prov and OSLC Traces

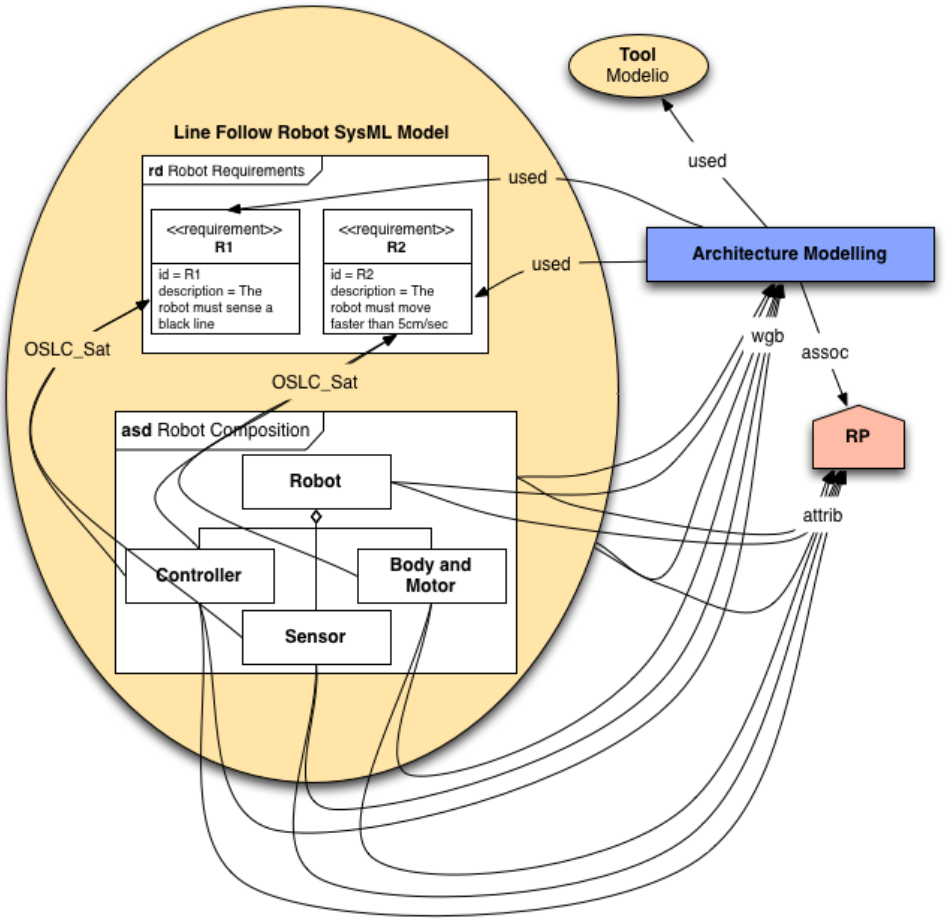
- `RobotSysML#1:Requirements.R1 : OSLC_Elab : UserDoc#1`
- `RobotSysML#1:Requirements.R2 : OSLC_Elab : UserDoc#1`
- `RobotSysML#1 : prov_attrib : Agent.RP`
- `RobotSysML#1:Requirements.Requirements_Diagram : prov_attrib : Agent.RP`
- `RobotSysML#1:Requirements.R1 : prov_attrib : Agent.RP`
- `RobotSysML#1:Requirements.R2 : prov_attrib : Agent.RP`
- `RobotSysML#1 : prov_wgb : Activity.Requirement_Engineering`
- `RobotSysML#1:Requirements.Requirements_Diagram : prov_wgb : Activity.Requirement_Engineering`
- `RobotSysML#1:Requirements.R1 : prov_wgb : Activity.Requirement_Engineering`
- `RobotSysML#1:Requirements.R2 : prov_wgb : Activity.Requirement_Engineering`
- `Activity.Requirement_Management : prov_used : UserDoc#1`
- `Activity.Requirement_Management: prov_used : Tool.Modelio`
- `Activity.Requirement_Management : prov_assoc : Agent.RP`

D.2.4 Data Gathering and Archiving

1. SysML model saved to modelio working files.
2. Modelio queries the use for their ID.
3. SysML model exported to SysML/<modelname>.zip. This file is committed to GIT, Modelio records the version number.
4. Modelio queries the user to indicate which diagrams/submodels have been updated/created.
5. Modelio creates a list of tracebility links based upon the new requirements added and their origin properties
6. Modelio creates the provenance relations
7. The traceability and provenance relations, in OSLC triples, are sent to the traceability Daemon.

D.3 Requirements → SysML Model ASD

D.3.1 Element Graph



D.3.2 Textual Description

The next step concerns the *Architectural Modelling* activity which uses the *Modelio* tool to create a new version of the SysML robot model. Requirements *R1* and *R2* are used by this activity to generate the *SysML elements in a SysML Model*: an *Architecture Structure Diagram*, the *Robot* system; a *Controller* cyber component; a *Sensor* physical component and a *Body and Motor* physical component. These SysML components in turn satisfy the requirements: the *Controller* and *Sensor* components satisfy requirement *R1* and the *Controller Body and Motor* components satisfy requirement *R2*. The *Architectural Modelling* activity is associated with the agent *RP*, to whom the new version of the SysML model, the *Architecture Structure Diagram*, and *Robot*, *Controller*, *Sensor* and *Body and Motor* SysML elements are attributed.

D.3.3 Prov and OSLC Traces

- `RobotSysML#2:Architecture/Controller : OSLC_Sat : RobotSysML#1:Requirements.R1`
- `RobotSysML#2:Architecture/Sensor : OSLC_Sat : RobotSysML#1:Requirements.R1`
- `RobotSysML#2:Architecture/Controller : OSLC_Sat : RobotSysML#1:Requirements.R2`
- `RobotSysML#2:Architecture/Body_and_Motor : OSLC_Sat : RobotSysML#1:Requirements.R2`

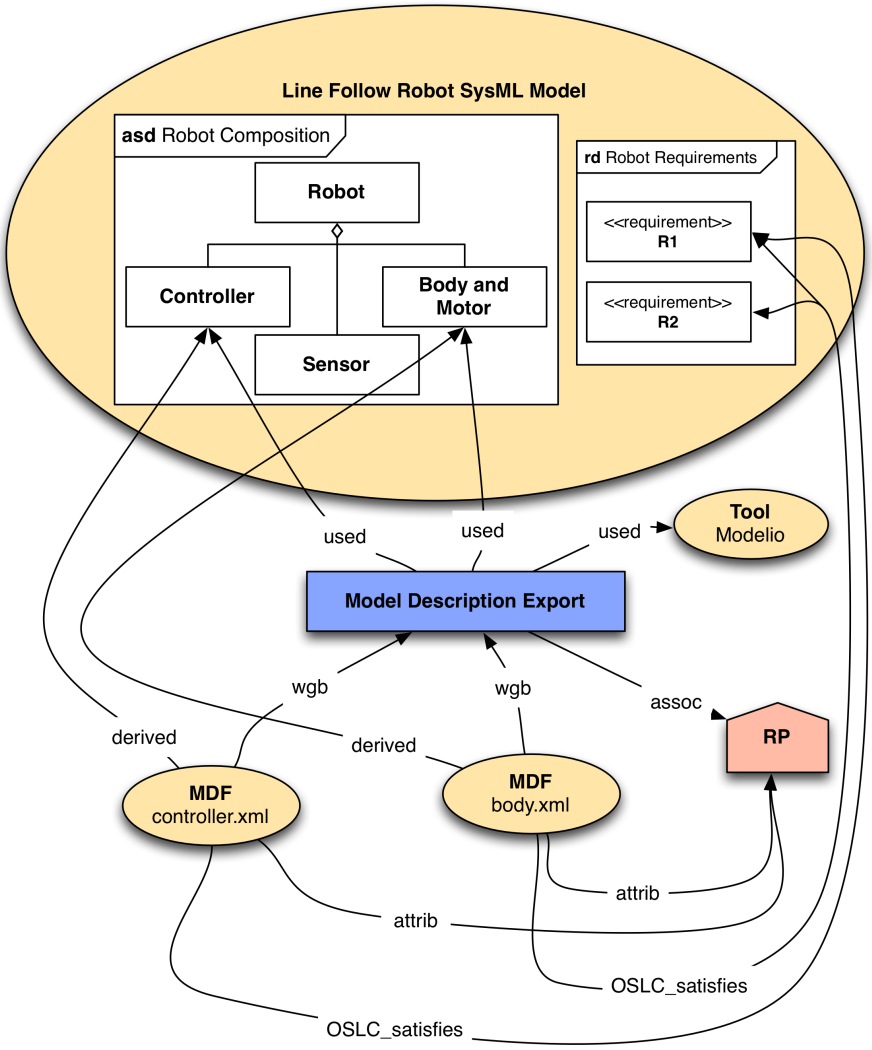
- RobotSysML#2 : prov_attrib : Agent.RP
- RobotSysML#2:Architecture/Controller : prov_attrib : Agent.RP
- RobotSysML#2:Architecture/Sensor : prov_attrib : Agent.RP
- RobotSysML#2:Architecture/Robot : prov_attrib : Agent.RP
- RobotSysML#2:Architecture/Body_And_Motor : prov_attrib : Agent.RP
- RobotSysML#2:Architecture/ASD : prov_attrib : Agent.RP
- RobotSysML#2 : prov_wgb : Activity.Architecture_Modelling
- RobotSysML#2:Architecture/Controller : prov_wgb : Activity.Architecture_Modelling
- RobotSysML#2:Architecture/Sensor : prov_wgb : Activity.Architecture_Modelling
- RobotSysML#2:Architecture/Robot : prov_wgb : Activity.Architecture_Modelling
- RobotSysML#2:Architecture/Body_And_Motor : prov_wgb : Activity.Architecture_Modelling
- RobotSysML#2:Architecture/ASD : prov_wgb : Activity.Architecture_Modelling
- Activity.Architecture_Modelling : prov_used : RobotSysML#1:Requirements.R1
- Activity.Architecture_Modelling : prov_used : RobotSysML#1:Requirements.R2
- Activity.Architecture_Modelling : prov_used : Tool.Modelio
- Activity.Architecture_Modelling : prov_assoc : Agent.RP

D.3.4 Data Gathering and Archiving

1. SysML model saved to Modelio working files
2. Modelio queries user for their ID
3. Modelio queries local GIT for previous version number of SysML/<modelname>.zip
4. SysML model exported to SysML/<modelname>.zip. This file is committed to GIT, Modelio records the new version number.
5. Modelio queries the user to indicate which diagrams/submodels have been updated/created.
6. Modelio creates traceability links based upon the new/modified elements and reading in the URIs of the requirements elements
7. Modelio creates provenance links new/modified elements and reading in the URIs of the requirements elements
8. The traceability and provenance relations, in OSLC triples, are sent to the traceability Daemon.

D.4 SysML Model ASD → Model Description Files

D.4.1 Element Graph



D.4.2 Textual Description

The *Model Description Export* activity uses the *Modelio* tool. The *Controller* and *Body and Motor* SysML entities are used to generate *controller.xml* and *body.xml* model description files. This *Model Description Export* activity is associated with the agent *RP*, to whom the *controller.xml* and *body.xml* model description files are attributed. The model description files satisfy the requirements *R1* and *R2*.

D.4.3 Prov and OSLC Traces

- `controller.xml#1 : prov_derived : RobotSysML#2:Architecture/Controller`
- `body.xml#1 : prov_derived : RobotSysML#2:Architecture/Body_and_Motor`
- `controller.xml#1 : prov_attrib : Agent.RP`
- `body.xml#1 : prov_attrib : Agent.RP`
- `controller.xml#1 : prov_wgb : Activity.Model_Description_Export`

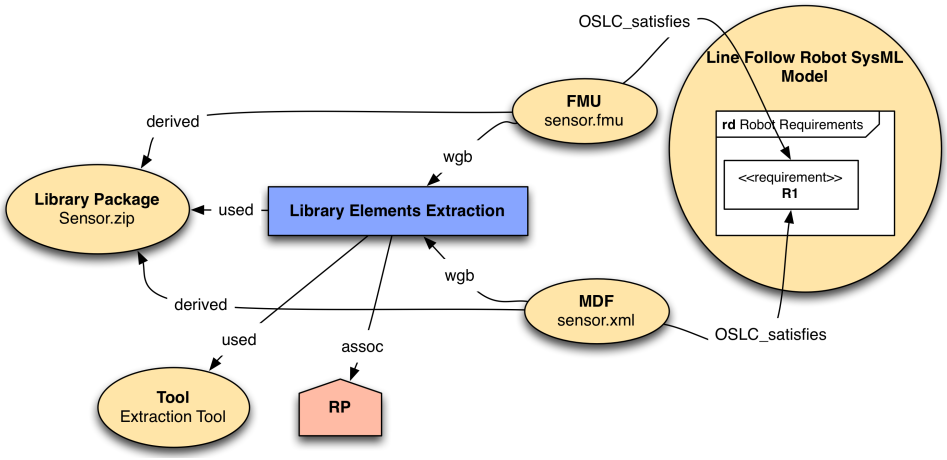
- `body.xml#1 : prov_wgb : Activity.Model_Description_Export`
- `controller.xml#1 : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `body.xml#1 : OSLC_satisfies : RobotSysML#1:Requirements.R2`
- `Activity.Model_Description_Export : prov_used : RobotSysML#2:Architecture/Controller`
- `Activity.Model_Description_Export : prov_used : RobotSysML#2:Architecture/Body_and_Motor`
- `Activity.Model_Description_Export : prov_used : Tool.Modelio`
- `Activity.Model_Description_Export : prov_assoc : Agent.RP`

D.4.4 Data Gathering and Archiving

1. Exported model description files saved to MDFs/body.xml and MDFs/controller.xml
2. Modelio queries user for their ID
3. MDFs/body.xml and MDFs/controller.xml are committed to GIT and their version numbers recorded
4. Modelio creates provenance relations based upon the URIs
5. Modelio creates OSLC triples and sends them to the traceability daemon

D.5 Library Sensor → Sensor Model Description File

D.5.1 Element Graph



D.5.2 Textual Description

The *Library Elements Extraction* activity uses a *Sensor.zip* library package to generate *sensor.fmu* and *sensor.xml* entities. These entities are therefore derived from the library package. The activity uses an *Extraction Tool* and is associated with the agent *RP*. The model description and FMU files satisfies the requirement *R1*.

D.5.3 Prov and OSLC Traces

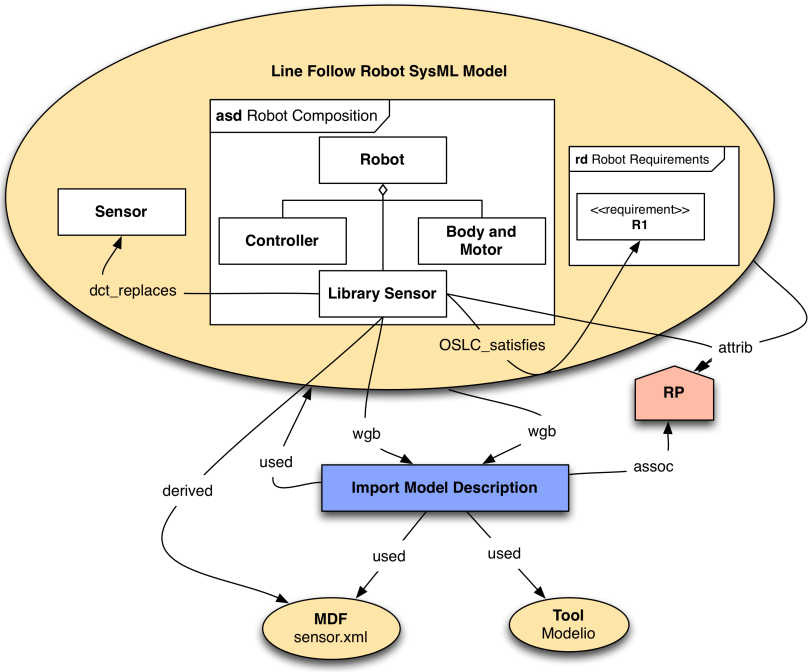
- `sensor.fmu#1 : prov_derived : Sensor.zip#1`
- `sensor.xml#1 : prov_derived : Sensor.zip#1`
- `sensor.fmu#1 : prov_wgb : Activity.Library_Elements_Extraction`
- `sensor.xml#1 : prov_wgb : Activity.Library_Elements_Extraction`
- `sensor.fmu#1 : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `sensor.xml#1 : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `Activity.Library_Elements_Extraction : prov_used : Sensor.zip#1`
- `Activity.Library_Elements_Extraction : prov_used : Tool.Extraction_Tool`
- `Activity.Library_Elements_Extraction : prov_assoc : Agent.RP`

D.5.4 Data Gathering and Archiving

1. The sensor.fmu and sensor.xml file are extracted by the extraction tool to FMUs/sensor.fmu and MDFs/sensor.xml
2. The app queries user for their ID
3. The app queries for latest versions of requirements, user is able to tag sensor.fmu and sensor.xml with the appropriate requirements
4. FMUs/sensor.fmu and MDFs/sensor.xml are committed to GIT and their version numbers recorded
5. The app creates provenance relations based upon the URIs
6. The app creates OSLC triples and sends them to the traceability daemon

D.6 Model Description Files Import → SysML model ASD

D.6.1 Element Graph



D.6.2 Textual Description

The *Import Model Description* activity uses the *Modelio* tool to create a new version of the SysML robot model. The activity uses the *sensor.xml* model description file to generate the *Library Sensor* component entity in the new SysML model, which is added to the SysML *Architecture Structure Diagram* entity, replacing the old *Sensor* component, and satisfies the requirement *R1*. . This *Import Model Description* activity is associated with the agent *RP*, and the new SysML model and *Library Sensor* component entity is attributed to *RP*.

D.6.3 Prov and OSLC Traces

- `RobotSysML#3 : prov_wgb : Activity.Import_Model_Description`
- `RobotSysML#3 : prov_attrib : Agent.RP`
- `RobotSysML#3:Architecture/Controller : prov_derived : RobotSysML#2:Architecture/Controller`
- `RobotSysML#3:Architecture/Controller : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `RobotSysML#3:Architecture/Controller : prov_attrib : Agent.RP`
- `RobotSysML#3:Architecture/Body_and_Motor : prov_derived : RobotSysML#2:Body_and_Motor`
- `RobotSysML#3:Architecture/Body_and_Motor : OSLC_satisfies : RobotSysML#1:Requirements.R2`
- `RobotSysML#3:Architecture/Body_and_Motor : prov_attrib : Agent.RP`
- `RobotSysML#3:Architecture/Library_Sensor : prov_attrib : Agent.RP`
- `RobotSysML#3:Architecture/Library_Sensor : prov_derived : sensor.xml#1`

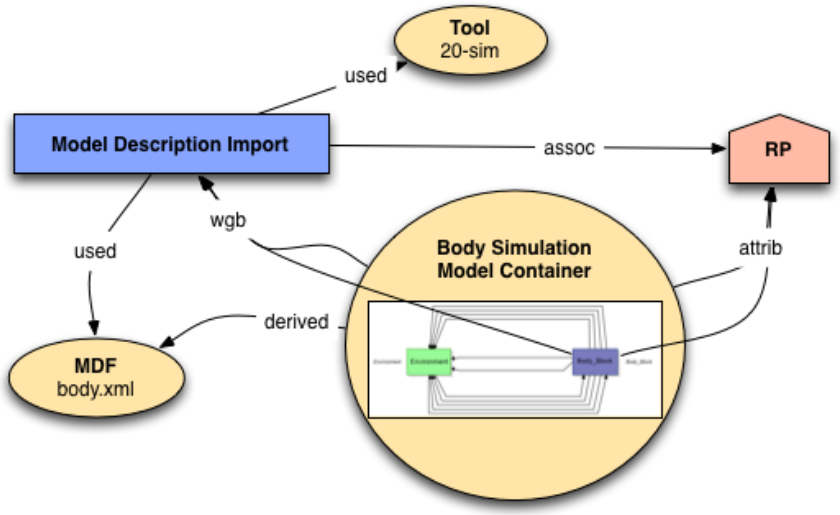
- `RobotSysML#3:Architecture/Library_Sensor : prov_wgb : Activity.Import_Model_Description`
- `RobotSysML#3:Architecture/Library_Sensor : dct_replaces : RobotSysML#3:Architecture/Library_Sensor`
- `RobotSysML#3:Architecture/Library_Sensor : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `Activity.Import_Model_Description : prov_used : RobotSysML#2`
- `Activity.Import_Model_Description : prov_used : sensor.xml#1`
- `Activity.Import_Model_Description : prov_used : Tool.Modelio`
- `Activity.Import_Model_Description : prov_assoc : Agent.RP`

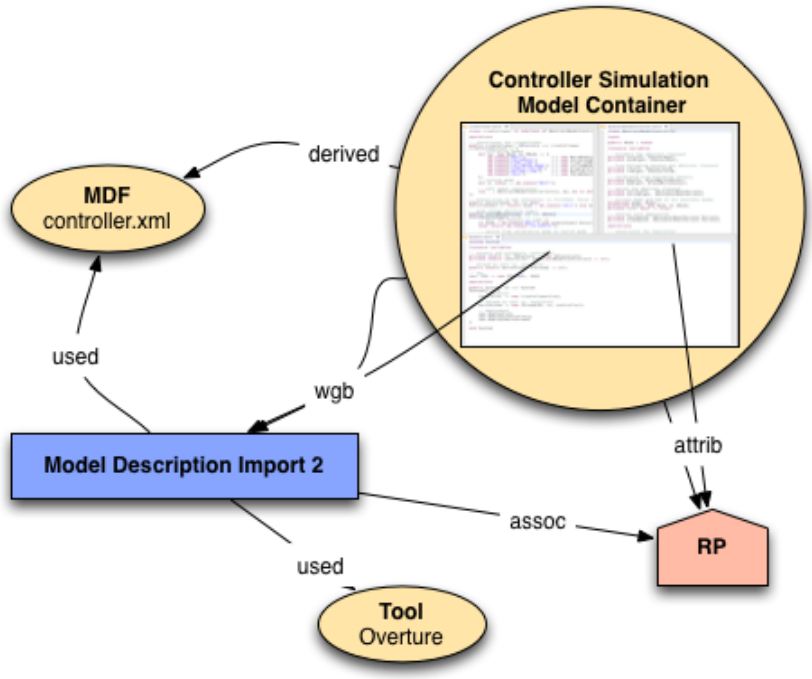
D.6.4 Data Gathering and Archiving

1. Modelio queries GIT to obtain the current version number of the SysML model.
2. SysML model saved and exported to SysML/<modelname>.zip
3. The app queries user for their ID
4. SysML/<modelname>.zip is committed to GIT and the new version recorded
5. The app creates provenance relations based upon the URIs
6. The app creates OSLC triples and sends them to the traceability daemon

D.7 Model Description Files → Simulation Models

D.7.1 Element Graph





D.7.2 Textual Description

In this example, there are two instances of the model description import activity. In this text, we refer to only one, as they are broadly similar. The *Model Description Import* activity uses the *20-sim* tool and the *body.xml* model description files to generate a *Body Simulation Model Container* and skeleton model element – in this case a 20-sim *Body* block. This model container and skeleton models are therefore derived from the *body.xml* model description file and attributed to the agent *RP*.

D.7.3 Prov and OSLC Traces

- `BodySimModel#1 : prov_derived : body.xml#1`
- `BodySimModel#1 : prov_wgb : Activity.Model_Description_Import#1`
- `BodySimModel#1 : prov_attrib : Agent.RP`
- `BodySimModel#1.Body_Block : prov_wgb : Activity.Model_Description_Import#1`
- `BodySimModel#1.Body_Block : prov_attrib : Agent.RP`
- `Activity.Model_Description_Import#1 : prov_used : body.xml#1`
- `Activity.Model_Description_Import#1 : prov_used : Tool.20-sim`
- `Activity.Model_Description_Import#1 : prov_assoc : Agent.RP`

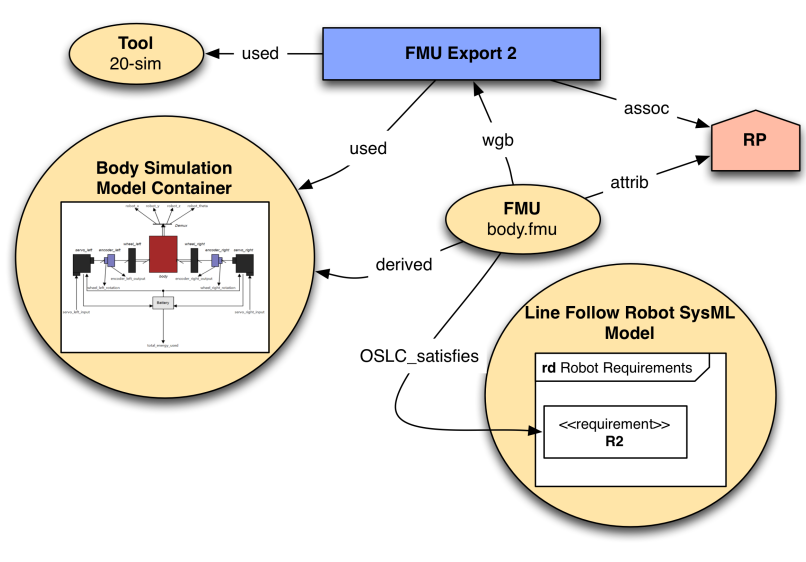
- `ControllerSimModel#1 : prov_derived : controller.xml#1`
- `ControllerSimModel#1 : prov_wgb : Activity.Model_Description_Import#2`
- `ControllerSimModel#1 : prov_attrib : Agent.RP`
- `ControllerSimModel#1.ControllerClass : prov_wgb : Activity.Model_Description_Import#2`
- `ControllerSimModel#1.ControllerClass : prov_attrib : Agent.RP`
- `Activity.Model_Description_Import#2 : prov_used : controller.xml#1`
- `Activity.Model_Description_Import#2 : prov_used : Tool.Overture`
- `Activity.Model_Description_Import#2 : prov_assoc : Agent.RP`

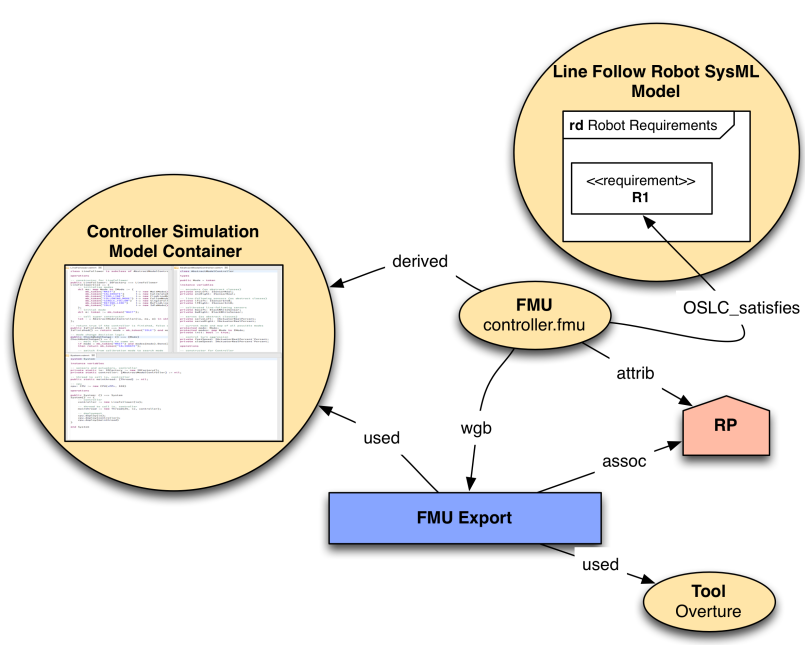
D.7.4 Data Gathering and Archiving

1. Overture/20-sim query GIT to obtain the current version appropriate model description file.
2. Overture/20-sim save their respective models
3. The app queries user for their ID
4. Models are committed to GIT, version numbers are recorded
5. Overture/20-sim query the user to identify sub-models to be traced
6. Overture/20-sim create provenance and traceability relations based upon the URIs
7. Traceability data is sent to the traceability daemon

D.8 Simulation Models → FMUs

D.8.1 Element Graph





D.8.2 Textual Description

As with the previous step, there are two instances of the FMU export activity. The *FMU Export* activity used a completed *Body Simulation Model Container* entity and the *20-sim* tool to generate a *body.fmu* FMU file. The FMU satisfies requirement *R2*. The activity is associated with the agent *RP*, and the *body.fmu* FMU file is attributed with the agent *RP*. The *FMU Export 2* activity is largely the same, from the perspective of using the *Overture* tool with the robot controller.

D.8.3 Prov and OSLC Traces

- `body.fmu#1 : prov_derived : BodySimModel#1.BodyBlock`
- `body.fmu#1 : prov_wgb : Activity.FMU_Export#1`
- `body.fmu#1 : prov_attrib : Agent.RP`
- `body.fmu#1 : OSLC_satisfies : RobotSysML#1:Requirements.R2`
- `Activity.FMU_Export#1 : prov_used : BodySimModel#1.BodyBlock`
- `Activity.FMU_Export#1 : prov_used : Tool.20-sim`
- `Activity.FMU_Export#1 : prov_assoc : Agent.RP`

- `controller.fmu#1 : prov_derived : ControllerSimModel#1.ControllerClass`
- `controller.fmu#1 : prov_wgb : Activity.FMU_Export#2`
- `controller.fmu#1 : prov_attrib : Agent.RP`
- `controller.fmu#1 : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `Activity.FMU_Export#2 : prov_used : ControllerSimModel#1.ControllerClass`
- `Activity.FMU_Export#2 : prov_used : Tool.Overture`
- `Activity.FMU_Export#2 : prov_assoc : Agent.RP`

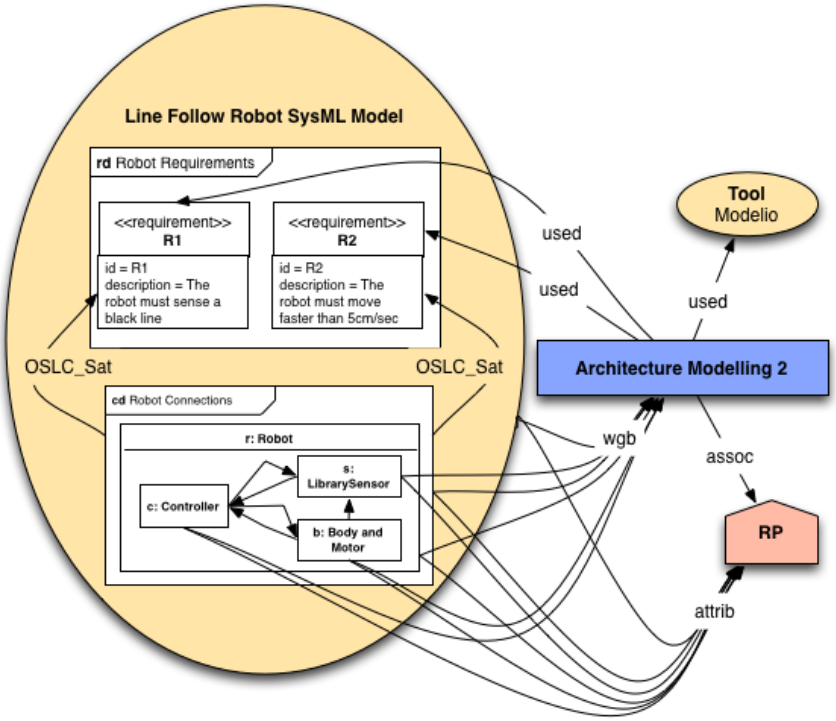
D.8.4 Data Gathering and Archiving

1. Overture/20-sim export their respective FMUs

2. Overture/20-sim query user for their ID
3. FMUs are committed to GIT, version numbers are recorded
4. Overture/20-sim create provenance and traceability relations based upon the URIs
5. Traceability data is sent to the traceability daemon

D.9 Create SysML Model Connections Diagram

D.9.1 Element Graph



D.9.2 Textual Description

The *Architecture Modelling 2* activity defines a connections diagram in a new version of the *Line follow Robot SysML Model*. It should be noted that this activity could be performed earlier. The activity uses requirements *R1* and *R2*, and the *Modelio* tool; generating a new version of the *Line follow Robot SysML Model* with a connections diagram and block instances *r*, *c*, *s* and *b*, attributed to agent *RP*. The connection diagram satisfies requirements due to using existing component definitions.

D.9.3 Prov and OSLC Traces

- `RobotSysML#4:Architecture/CD : OSLC_satisfies : RobotSysML#1:Requirements.R1`
- `RobotSysML#4:Architecture/CD : OSLC_satisfies : RobotSysML#1:Requirements.R2`
- `RobotSysML#4 : prov_wgb : Activity.Architecture_Modelling#2`
- `RobotSysML#4:Architecture/CD : prov_wgb : Activity.Architecture_Modelling#2`
- `RobotSysML#4:Architecture/r : prov_wgb : Activity.Architecture_Modelling#2`
- `RobotSysML#4:Architecture/r.c : prov_wgb : Activity.Architecture_Modelling#2`

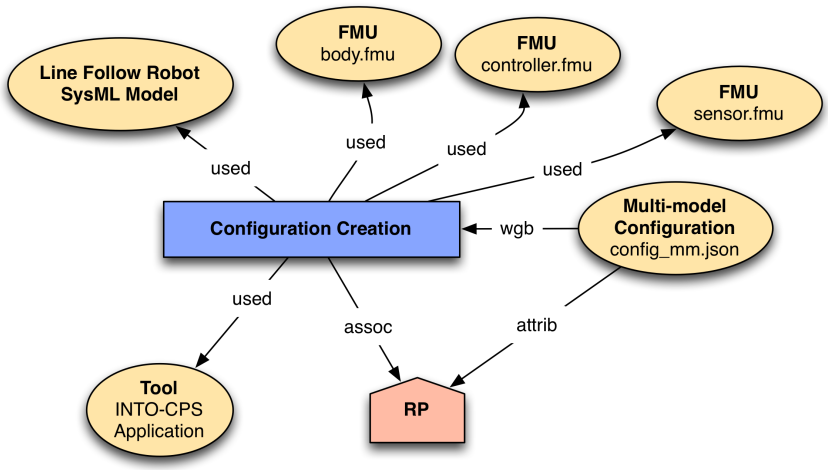
- RobotSysML#4:Architecture/r.s : prov_wgb : Activity.Architecture_Modelling#2
- RobotSysML#4:Architecture/r.b : prov_wgb : Activity.Architecture_Modelling#2
- RobotSysML#4 : prov_attrib : Agent.RP
- RobotSysML#4:Architecture/CD : prov_attrib : Agent.RP
- RobotSysML#4:Architecture/r : prov_attrib : Agent.RP
- RobotSysML#4:Architecture/r.c : prov_attrib : Agent.RP
- RobotSysML#4:Architecture/r.s : prov_attrib : Agent.RP
- RobotSysML#4:Architecture/r.b : prov_attrib : Agent.RP
- Activity.Architecture_Modelling#2 : prov_used : RobotSysML#3:Requirements.R1
- Activity.Architecture_Modelling#2 : prov_used : RobotSysML#3:Requirements.R2
- Activity.Architecture_Modelling#2 : prov_used : Tool.Modelio
- Activity.Architecture_Modelling#2 : prov_assoc : Agent.RP

D.9.4 Data Gathering and Archiving

1. Modelio queries the local GIT to obtain the current version of the SysML model
2. Modelio saves the SysML model and exports the .zip version for archiving
3. Modelio queries the user for their ID
4. The zip version of the model is committed to GIT and the version number recorded
5. Modelio creates provenance and traceability relations based upon the URIs
6. Traceability data is sent to the traceability daemon

D.10 SysML Model CD → Simulation Configuration

D.10.1 Element Graph



D.10.2 Textual Description

The *Configuration Creation* activity uses the *Line follow Robot SysML Model* (in particular the block definitions, block instances and their connections defined in the *Architecture Structure Diagram* and *Connections Diagram*), two new FMUs (*body.fmu* and

controller.fmu), the library *sensor.fmu.* and the *INTO-CPS Application* to generate the *config_mm.json* Multi-model Configuration entity. This *config_mm.json* contains details of the FMUs to use, the connections between the FMUs and any shared design parameters.

D.10.3 Prov and OSLC Traces

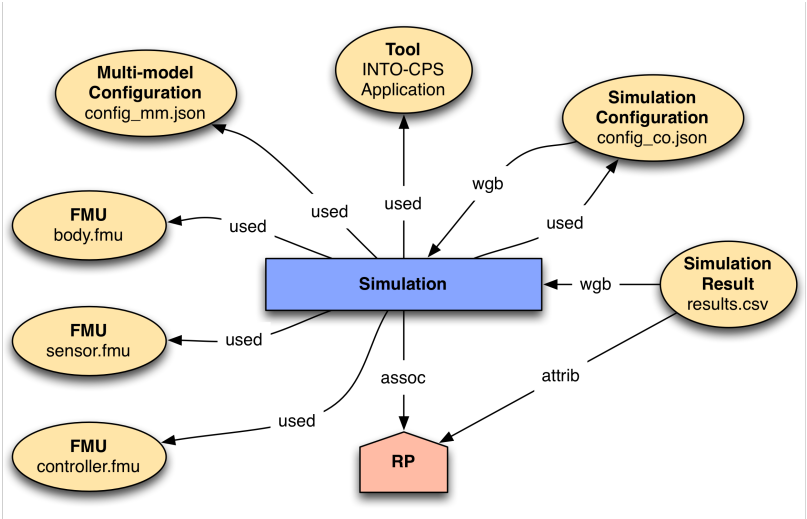
- `config_mm.json#1 : prov_wgb : Activity.Configuration_Creation`
- `config_mm.json#1 : prov_attrib : Agent.RP`
- `Activity.Configuration_Creation: prov_used : RobotSysML#4`
- `Activity.Configuration_Creation: prov_used : Body.FMU#1`
- `Activity.Configuration_Creation: prov_used : Controller.FMU#1`
- `Activity.Configuration_Creation: prov_used : Sensor.FMU#1`
- `Activity.Configuration_Creation : prov_used : Tool.INTO-CPS_Application`
- `Activity.Configuration_Creation : prov_assoc : Agent.RP`

D.10.4 Data Gathering and Archiving

1. The INTO-CPS application queries the local GIT to obtain the current version of the SysML model and the versions of the FMUs used to generate the configuration
2. The INTO-CPS application saves the multi-model configuration
3. The INTO-CPS application queries the user for their ID
4. The multi-model configuration is committed to GIT and the version number recorded
5. The INTO-CPS application creates provenance and traceability relations based upon the URIs
6. Traceability data is sent to the traceability daemon

D.11 Simulation Configuration and FMUs → Simulation Result

D.11.1 Element Graph



D.11.2 Textual Description

The *Simulation* activity uses the *config_ mm.json* Multi-model and Simulation Configuration entities; the collection of FMUs: *body.fmu*, *sensor.fmu* and *controller.fmu*; and the *INTO-CPS Application* tool. The activity generates the *results.csv* simulation results which are attributed to the agent *RP*.

D.11.3 Prov and OSLC Traces

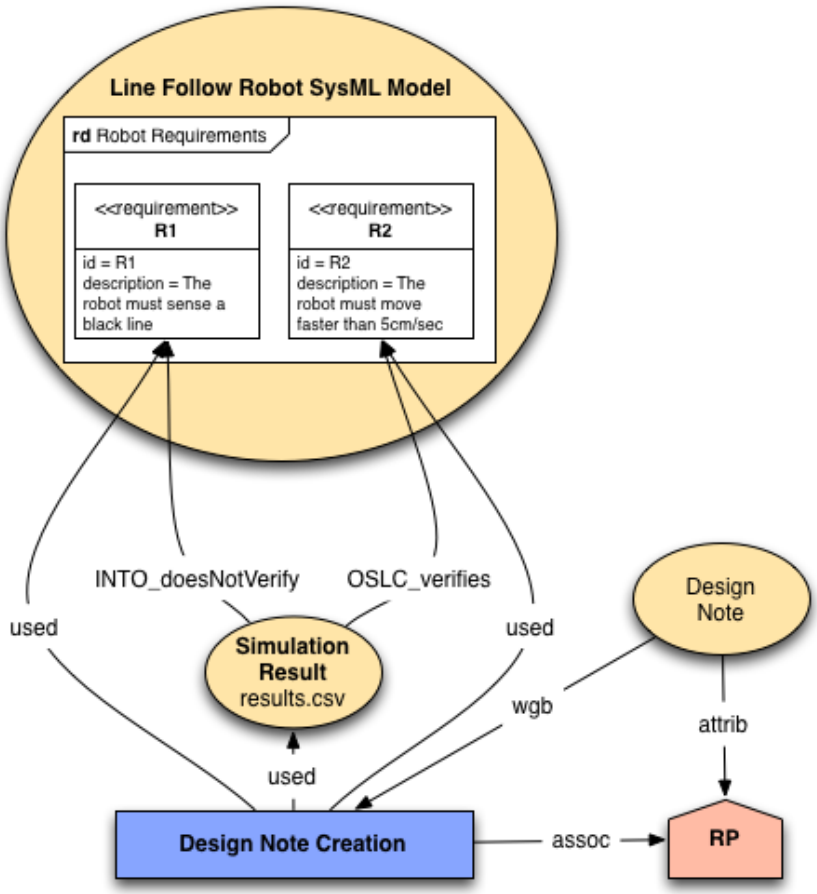
- `results.csv#1 : prov_wgb : Activity.Simulation`
- `config_mm.json#1 : prov_wgb : Activity.Simulation`
- `results.csv#1 : prov_attrib : Agent.RP`
- `Activity.Simulation : prov_used : config_mm.json#1`
- `Activity.Simulation : prov_used : body.fmu#1`
- `Activity.Simulation : prov_used : sensor.fmu#1`
- `Activity.Simulation : prov_used : controller.fmu#1`
- `Activity.Simulation : prov_used : Tool.INTO-CPS_Application`
- `Activity.Simulation : prov_assoc : Agent.RP`

D.11.4 Data Gathering and Archiving

1. The INTO-CPS application queries the local GIT to obtain the current version of Multi-model configuration and the FMUs used
2. The INTO-CPS application queries the user for final simulation parameters and saves the simulation configuration file
3. The simulation is then run and, if the user deems it to be a useful result the data gathering and archiving continues
4. The INTO-CPS application queries the user for their ID
5. The simulation configuration and simulation result are committed to GIT and the version numbers recorded
6. The INTO-CPS application creates provenance and traceability relations based upon the URIs
7. Traceability data is sent to the traceability daemon

D.12 Simulation Result and Requirements → Design Note

D.12.1 Element Graph



D.12.2 Textual Description

The *Design Note Creation* activity uses the *results.csv* simulation results and requirements *R1* and *R2* in the Robot SysML model. The *results.csv* results verify that requirement *R2* holds and that the *results.csv* results do not verify requirement *R2*. The activity, associated with agent *RP*, generates a *Design Note* that is attributed to agent *RP*.

D.12.3 Prov and OSLC Traces

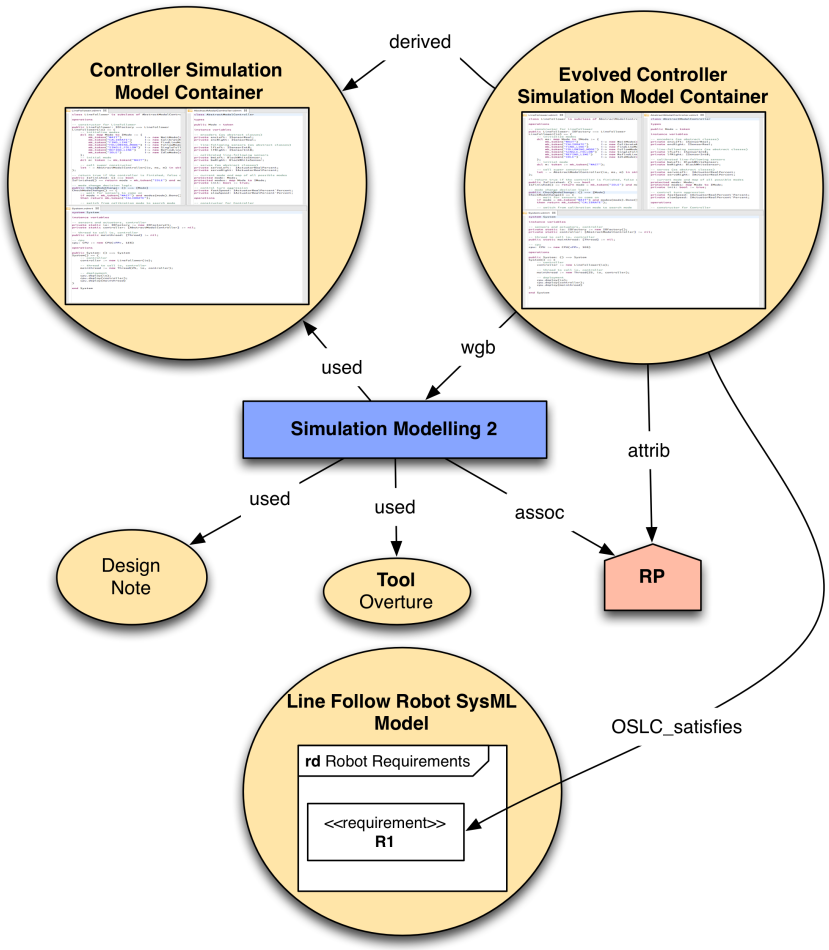
- `Design_Note#1 : prov_wgb : Activity.Design_Note_Creation`
- `Design_Note#1 : prov_attrib : Agent.RP`
- `Activity.Design_Note_Creation : prov:used : RobotSysML#5:Requirements.R1`
- `Activity.Design_Note_Creation : prov:used : RobotSysML#5:Requirements.R2`
- `Activity.Design_Note_Creation : prov_used : results.csv#1`
- `results.csv#1 : OSLC_verifies : RobotSysML#5:Requirements.R2`
- `results.csv#1 : INTO_doesNotVerify : RobotSysML#5:Requirements.R1`
- `Activity.Design_Note_Creation : prov_assoc : Agent.RP`

D.12.4 Data Gathering and Archiving

1. The user creates a design note, which is the generic term for any document other than a model that we wish to store and registers it with the INTO-CPS application, the design note is committed to GIT
2. The INTO-CPS application queries the GIT for the version number of the design note
3. The INTO-CPS application allows the user to find the version numbers of entities they want to relate to the design note for traceability, in this case these are the URIs for the requirements and simulation results.
4. The INTO-CPS application creates provenance and traceability relations based upon the URIs
5. Traceability data is sent to the traceability daemon

D.13 Design Note and Controller Model → Evolved Controller Model

D.13.1 Element Graph



D.13.2 Textual Description

Due to the previous activity identifying that the multi-model does not meet requirement *R2*, the*Simulation Modelling 2* activity evolves the model in the *Controller Simulation Model Container* entity; generating a new *Evolved Controller Simulation Model Container* derived form the first (we refer to version n and $n+1$ in the traces to signify the version changes. This evolved model satisfies requirement *R1*. The activity uses the *Design Note* and the *Overture* tool, and is associated with the agent *RP*. The new *Evolved Controller Simulation Model Container* is attributed to the agent *RP*.

D.13.3 Prov and OSLC Traces

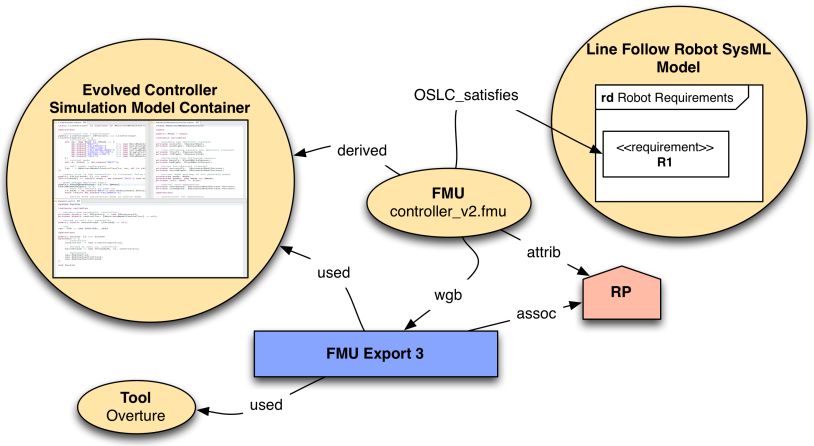
- `ControllerSimModel#2 : prov_wgb : ControllerSimModel#2`
- `ControllerSimModel#2 : prov_attrib : Agent.RP`
- `ControllerSimModel#2 : prov_derived : ControllerSimModel#1`
- `ControllerSimModel#2 : OSLC_satisfies : RobotSysML#5:Requirements.R1`
- `Activity.Simulation_Modelling#2 : prov_used : ControllerSimModel#1`
- `Activity.Simulation_Modelling#2 : prov_used : Design_Note#1`
- `Activity.Simulation_Modelling#2 : prov_used : Tool.Overture`
- `Simulation_Modelling#2 : prov_assoc : Agent.RP`

D.13.4 Data Gathering and Archiving

1. Once the new version of the controller is produced in response to the design note, the user saves the model
2. Overture queries the GIT for the currently committed version of the controller model and records this value
3. The new controller model is committed and the new version recorded
4. Overture queries the user for their ID, they are also permitted to add a reference to the design note by retrieving its version from GIT
5. Overture constructs the traceability and provenance triples
6. Traceability data is sent to the traceability daemon

D.14 Evolved Controller Model → Evolved Controller FMU

D.14.1 Element Graph



D.14.2 Textual Description

The *FMU Export 3* activity used the new *Evolved Controller Simulation Model Container* entity and the *Overture* tool to generate a new *controller_ v2.fmu* FMU file, which satisfies requirement *R1*. The activity is associated with the agent *RP*, and the *controller_ v2.fmu* FMU file is attributed with the agent *RP*.

D.14.3 Prov and OSLC Traces

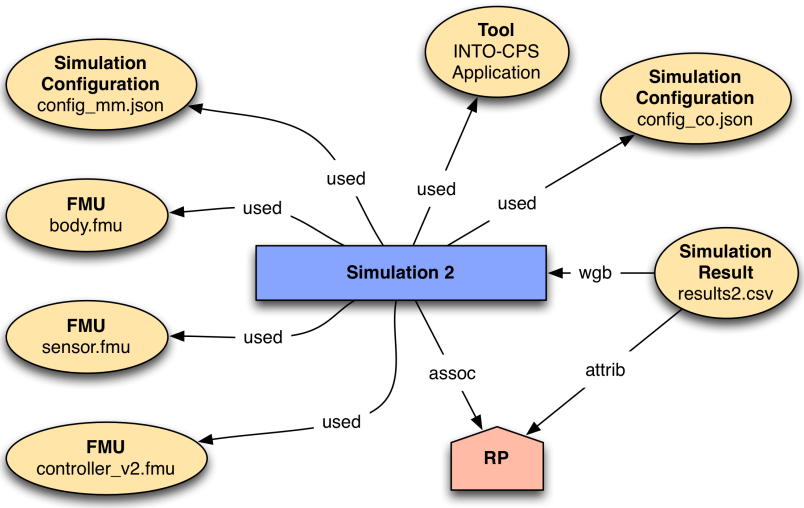
- `controller.fmu#2 : prov_derived : ControllerSimModel#n+1`
- `controller.fmu#2 : prov_wgb : Activity.FMU_Export#1`
- `controller.fmu#2 : prov_attrib : Agent.RP`
- `controller.fmu#2 : OSLC_satisfies : RobotSysML#5:Requirements.R1`
- `Activity.FMU_Export#3 : prov_used : ControllerSimModel#n+1`
- `Activity.FMU_Export#3 : prov_used : Tool.Overture`
- `Activity.FMU_Export#3 : prov_assoc : Agent.RP`

D.14.4 Data Gathering and Archiving

1. Overture queries the GIT for the currently committed version of the controller FMU and records this value
2. Overture exports the new controller FMU
3. The new controller FMU is committed and the new version recorded
4. Overture queries the user for their ID
5. Overture constructs the traceability and provenance triples
6. Traceability data is sent to the traceability daemon

D.15 New Config and FMUs → New Simulation Result

D.15.1 Element Graph



D.15.2 Textual Description

Given the evolved system design, the *Simulation 2* activity uses the *config_mm.json* Multi-model Configuration entity; the collection of FMUs: *body.fmu*, *sensor.fmu* and evolved *controller_v2.fmu*; and the *INTO-CPS Application* tool. The activity generates a new *results_v2.csv* collection of simulation results which are attributed to the agent *RP*.

D.15.3 Prov and OSLC Traces

- `results.csv#2 : prov_wgb : Activity.Simulation#2`
- `results.csv#2 : prov_attrib : Agent.RP`
- `Activity.Simulation : prov_used : config_mm.json#1`
- `Activity.Simulation : prov_used : body.fmu#1`
- `Activity.Simulation : prov_used : sensor.fmu#1`
- `Activity.Simulation : prov_used : controller.fmu#2`
- `Activity.Simulation : prov_used : Tool.INTO-CPS_Application`
- `Activity.Simulation : prov_assoc : Agent.RP`

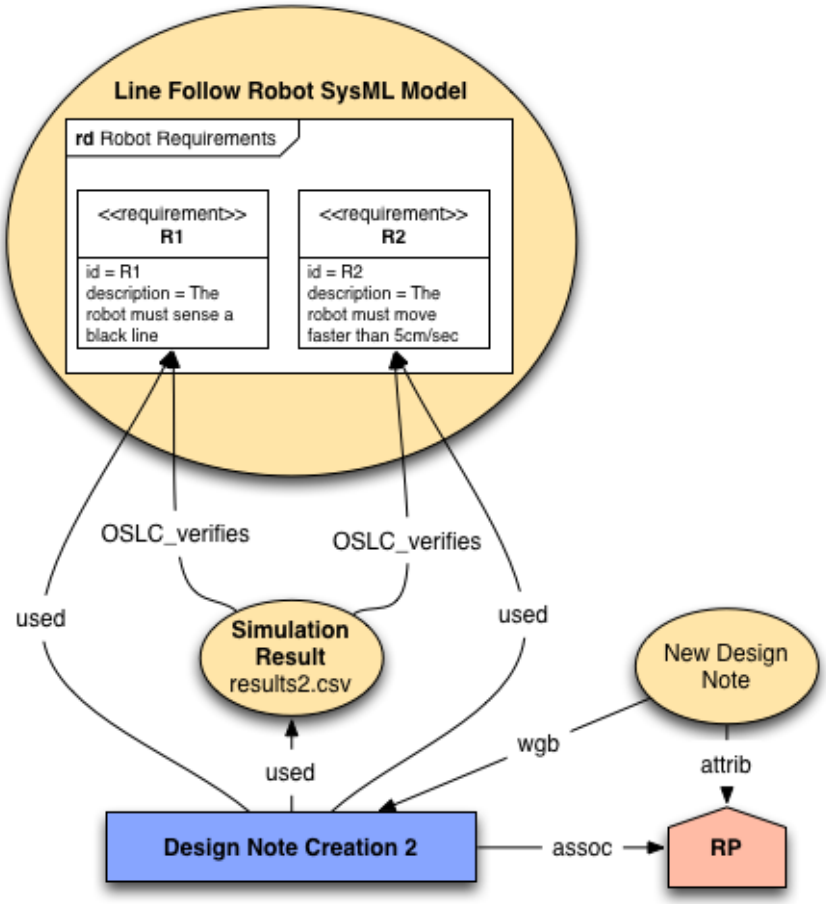
D.15.4 Data Gathering and Archiving

1. The INTO-CPS application queries the local GIT to obtain the current version of Multi-model configuration and the FMUs used
2. The INTO-CPS application queries the user for final simulation parameters and saves the simulation configuration file
3. The simulation is then run and, if the user deems it to be a useful result the data gathering and archiving continues
4. The INTO-CPS application queries the user for their ID

5. The simulation configuration and simulation result are committed to GIT and the version numbers recorded
6. The INTO-CPS application creates provenance and traceability relations based upon the URIs
7. Traceability data is sent to the traceability daemon

D.16 New Simulation Result and Requirements → New Design Note

D.16.1 Element Graph



D.16.2 Textual Description

The *Design Note Creation 2* activity uses the new *results_ v2.csv* simulation results and requirements *R1* and *R2* in the SysML robot model. The new design note states that the *results.csv* results verify that requirements *R1* and *R2* hold. The activity, associated with agent *RP*, generates a *Design Note 2* that is attributed to agent *RP*.

D.16.3 Prov and OSLC Traces

- `Design_Note#2 : prov_wgb : Activity.Design_Note_Creation#2`
- `Design_Note#2 : prov_attrib : Agent.RP`
- `Activity.Design_Note_Creation#2 : prov:used : RobotSysML#5:Requirements.R1`
- `Activity.Design_Note_Creation#2 : prov:used : RobotSysML#5:Requirements.R2`
- `Activity.Design_Note_Creation#2 : prov_used : results.csv#2`
- `results.csv#2 : OSLC_verifies : RobotSysML#5:Requirements.R2`
- `results.csv#2 : INTO_doesNotVerify : RobotSysML#5:Requirements.R1`
- `Activity.Design_Note_Creation#2 : prov_assoc : Agent.RP`

D.16.4 Data Gathering and Archiving

1. The user creates a design note, which is the generic term for any document other than a model that we wish to store and registers it with the INTO-CPS application, the design note is committed to GIT
2. The INTO-CPS application queries the GIT for the version number of the design note
3. The INTO-CPS application allows the user to find the version numbers of entities they want to relate to the design note for traceability, in this case these are the URIs for the requirements and simulation results.
4. The INTO-CPS application creates provenance and traceability relations based upon the URIs
5. Traceability data is sent to the traceability daemon