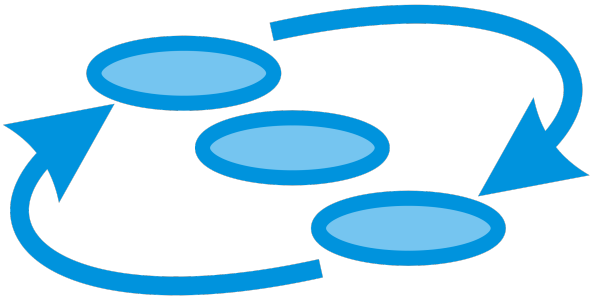
Grant Agreement: 644047

INtegrated TOol chain for model-based design of CPSs



INTO-CPS

Methods Progress Report 3

Deliverable Number: D3.3b

Version: 0.3

Date: 2017

Public Document

Contributors:

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

Editors:

Ken Pierce, UNEW

Reviewers:

Christian König, TWT
Etienne Brosse, ST
Frederik Foldager, 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	31-10-2017	Ken Pierce	Initial document
0.2	01-11-2017	Ken Pierce	Version for internal review
0.3	04-11-2017	Carl Gamble	MC/TA section of ontology updated

Abstract

This document reports progress in Work package 3 (Multi-modelling Methods) in the final year of INTO-CPS, covering workflows, design space exploration, traceability, guidelines, and pilot studies. It contains two appendices, presenting 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 the final year of INTO-CPS in Work Package 3. New and prospective users of the INTO-CPS technologies should refer to the Deliverable 3.3a Method Guidance 3 [FGP17].

Work Package 3 (Multi-modelling Methods) aims to provide pragmatic methods in the form of guidelines and patterns that support the 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. Guidance is presented in Method Guidance 3 (Deliverable 3.3a [FGP17]) and covers:

- A common concept base covering INTO-CPS terminology;
- Various ways to use the INTO-CPS tool chain in CPS development;
- Adapting requirements engineering processes to INTO-CPS;
- Architectural modelling in SysML;
- Using the machine-assisted traceability features of the tool chain;
- Initial multi-modelling using abstract FMUs;
- Modelling networks in FMI; and
- Design Space Exploration (DSE) analysis for multi-models.

These are supported by a comprehensive set of examples, described in Examples Compendium (Deliverable D3.6 [MGP+17]), which grew from the pilot studies work and serve as benchmarks for the methods and tools and as illustrations for the tool chain's capabilities.

This document outlines work undertaken in the five WP3 task towards guidelines (T3.1–T3.4) and examples (T3.5). It also presents technical work on traceability that underpins the tool implementation and guidelines that would be inappropriate for D3.3a. These are the traceability ontology in Appendix A and the traceability queries defined over this ontology in Appendix B.

2 Progress on WP3 Tasks

2.1 T3.1: Workflows

The focus of this task in the final year was to increase the influence of the user community on guidelines. To this end, training material was produced and deployed in several contexts, including:

- Training sessions for industrial partners in other projects;
- Tutorials for academic audiences at conferences;
- Masters students undertaking projects using INTO-CPS
- The INTO-CPS Summer School, including both industrial users and academics; and
- A post-doctoral researcher at UNEW applying the INTO-CPS technology to analyse the security of building management systems (BMSs), providing detailed feedback on guideline content.

Based on feedback from these various groups, potential improvements in the presentation of guidance were identified and integrated into Deliverable D3.3a [FGP17]. One improvement was to separate the material into introductory and advanced sections. The former providing new users with a clear "getting started" section, with the latter providing users who are comfortable with the overall approach further guidance in specific areas. Another improvement was the need for a unified description of the INTO-CPS profile focused on end-users.

This task also continued the "Tool Tsar" role established early Year 2, to ensure 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 D3.3a [FGP17], 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.

2.2 T3.2: Design Space Exploration

The focus of this task in the final year was to consolidate existing work on the DSE SysML profile, and look to explore the boundaries of DSE work which will be carried on in the INTO-CPS Association. The DSE SysML profile was extended with the ability to describe sweeping of multiple scenarios— where each combination of design parameters is tested against multiple, distinct test data. This task also worked closely with WP4 to ensure that the SysML profile descriptions and their realisation in the Modelio are aligned.

This task worked closely with WP1 to achieve DSE deployment within all four case studies. Additional effort (from outside the project) was leveraged to develop the ability to deploy DSE on cloud computing resources. Feasibility was demonstrated on an open source high-throughput computing framework (CONDOR). We plan to continue this work as part of the INTO-CPS Association to make cloud DSE services available to enterprises of all sizes.

2.3 T3.3: Provenance and Traceability

The focus of this task in the final year was to maintain the ontology in response to the evolving tools in the tool chain, to work with WP4/5 to ensure that the implemented traceability features align with the design, and to develop example queries over the ontology to be implemented in the INTO-CPS Application. Updates to the ontology focused on model checking and test automation, these are included in the latest version of the living ontology document in Appendix A.

Alignment with the tool implementation was achieved by testing the emerging traceability features against the reference example (reported previously in Deliverable D3.2b [FGPP16]). This feedback loop led to updates in the tool specification and implementation, as well as the guidance in Deliverable D3.3a [FGP17]. Queries were developed for the following use cases (and are given in Appendix B:

1. Impact analysis
   - Forward traceability (from requirements to entities)
   - Backwards traceability (from FMU to requirements)
   - Backwards traceability (from components to requirements)
2. Simulation sources
   - Find all simulations
   - Find sources and sinks for a simulation
3. Coverage
   - Requirements without architecture elements
   - Requirements without simulation models
   - Requirements without FMUs
   - Requirements without positive simulation results
   - Requirements without any simulation results
4. User impact
   - Find all users in the database
   - Find all artefacts influenced by a user
   - Find all activities performed by a user

2.4 T3.4: Guidelines

This work in this task covered three main aims: to establish and maintain a shared concepts base for the project, to ensure guidelines on various project technologies are accessible within the project, and to produce effort to write guidance the Method Guidance deliverables.

The majority of effort of this task this year fell at the end, producing guidance for Deliverable 3.3a [FGP17]. In particular this involved producing a new chapter on SysML, collating information from a wide range of project outputs into a single, user-focused chapter, and producing a chapter on traceability, describing how best to approach the tool chain to realise the benefits of the machine-assisted traceability features.

Additionally, in order to ensure that guidelines on various project technologies were available and accessible within the project (especially for industrial users), this task continued its series of webinars, hosted by TWT. It held webinars during the first half of the year on design space exploration (DSE) and traceability, following on from last year's webinars on concepts, co-simulation and workflows.

Finally, maintaining the concept base included adding new terms as required based upon new developments in the project, and ensuring any disagreements with existing terms are resolved. The concept base stabilised in the second year and did not need updating in the final year. The concept base is included in Deliverable 3.3a [FGP17], and was used to create a glossary for the project internally.

2.5 T3.5: Pilot Case Studies

This focus of this task in the final year of the project was to produce a stable set of examples, covering of all aspects of the tool chain. As described in the roadmap in Deliverable D3.5 [PGP+16], this required extension of the pilot studies to include code generation (CG), design space exploration (DSE), test automation (TA), model checking (MC), and SysML.

In addition to the Examples Compendium document, WP3 worked closely with WP4 to ensure that "README" descriptions of each example are available within the INTO-CPS Application, to be displayed when an example is imported. This included reframing text to the context of the INTO-CPS Application, including removal of diagrams (READMEs must be text-only).

Procedure for Accepting Examples into the Examples Compendium

In addition to building upon existing case studies within WP3, a procedure was defined for accepting examples from other work packages and challenge problems from outside including the Industrial Follower Group (IFG) and Academic Follower Group (AFG). Two examples were accepted through this process from project partners. While no challenges were accepted from the IFG or AFG, this process will remain as part of the INTO-CPS Association to help grow examples beyond the project lifetime.

1. Those wishing to submit an example should supply a brief description of the example should contact the following way:
   - Project partners contact WP3 leader;
   - AFG members contact the AFG coordinator; and
   - IFG members contact the IFG coordinator.
2. If deemed appropriate (we will consider for example, the uniqueness of the study, use of technologies, different domains and complexity), the WP3 group will make contact; candidate studies must include:
   (a) Pilot study overview
   (b) Source models where available and descriptions. If models are not available, a description of the FMI interface (inputs, outputs and parameters) is a minimum.
   (c) At least one multi-model with a description of the connections.
   (d) Usage instructions providing an overview of the pilot.
   (e) An optional collection of modelling and analysis activities (e.g. design space exploration, test automation, code generation).
   (f) For each artefact, suppliers must state the tool versions used, and platform(s) used to co-simulate/analyse.
3. Supplied pilots must be made available on the INTO-CPS GitHub repository – the above information and artefacts should be sent to the examples team.
4. Suppliers may be contacted for clarification.
5. Maintenance of these examples is not the responsibility of the INTO-CPS project. We will aim to compatibility perform tests using submitted pilots on new tool chain releases, and will inform submitters if issues are encountered. The project will make the pilot studies unavailable in the INTO-CPS application until issues are corrected.

Pilot Studies in Year 3

The following table shows the ten examples in the final Examples Compendium, Deliverable D3.6 [MGP+17] and indicates how they evolved in the final year. Two of the examples, the Bicycle and Mass-Spring Damper, were accepted following the process above.

Example	Year 3 Updates
Single-tank Water Tank	Additional CG (AU)
Three-tank Water Tank	Additional CG (AU)
Fan Coil Unit (FCU)	Additional CG (AU); TA and MC (UY)
Line-following Robot	Additional CG (AU); test automation and model checking (UY)
Turn Indicator	Traceability and MC (VSI)
UAV	Bug fixes
Ether	TA and MC (UY)
Swarm of UAV	TA and MC (UY)
Autonomous Vehicle	New example (AI)
Mass Spring Damper	New example demonstrating COE stability (AU)

3 Conclusions

In the final year of the project, WP3 worked to produce a set of outputs that will carry INTO-CPS beyond the life of the project. These are: a comprehensive guidelines document supporting all aspects of the technologies, including tutorials; a set of stable and well-documented examples supporting the guidance and tutorials; an updated traceability ontology to help those wishing to integrate their tools; and example traceability queries for tool developers and expert users. These will be passed to the INTO-CPS Association to be updated as the technologies continue to grow.

References

[FGP17] John Fitzgerald, Carl Gamble, and Ken Pierce. Method Guidelines 3. Technical report, INTO-CPS Deliverable, D3.3a, December 2017.

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

[KLN+17] Christian König, Kenneth Lausdahl, Peter Niermann, Jos Höll, Carl Gamble, Oliver Mölle, Etienne Brosse, Tom Bokhove, Luis Diogo Couto, and Adrian Pop. INTO-CPS Traceability Implementation. Technical report, INTO-CPS Deliverable, D4.3d, December 2017.

[MGP+17] Martin Mansfield, Carl Gamble, Ken Pierce, John Fitzgerald, Simon Foster, Casper Thule, and Rene Nilsson. Examples Compendium 3. Technical report, INTO-CPS Deliverable, D3.6, December 2017.

[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 Traceability Ontology

In this section we describe an ontology that forms the basis of the provenance and traceability data in INTO-CPS. This ontology was used as the specification for implementation of traceability within the tools, as reported in Deliverable D4.3d [KLN+17]. The ontology is presented as a collection of SysML diagrams, where the majority of the diagrams are centered around one or more activities that are supported by the INTO-CPS tool chain. The figures contain a collection of SysML blocks: blue elements represent an *activity*; yellow elements represent an *entity*; and orange elements represents an *agent*.

A.1 Requirements

Starting with requirements, Figure 1 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*. The *Requirements* correspond to the requirements defined in the *Requirement Definition View* as described in Deliverable D3.3a [FGP17]. 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 2, 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 D3.3a [FGP17].

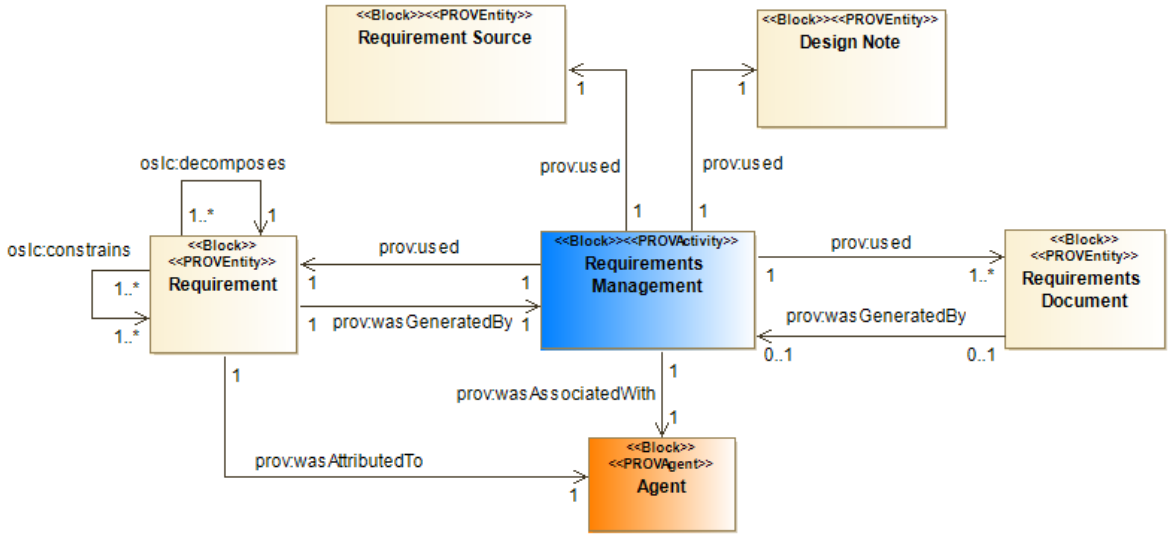


Figure 1: Block Definition Diagram (BDD) of the *Requirements Management* activity

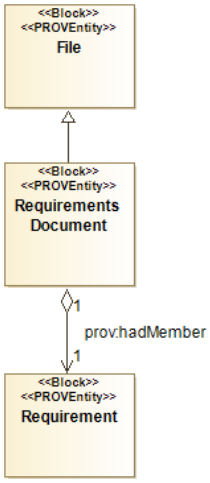


Figure 2: BDD of the *Requirements* files

A.2 Architecture Modelling

The activity of *Architecture Modelling* is presented in Figure 3. *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, as shown in Figure 4.

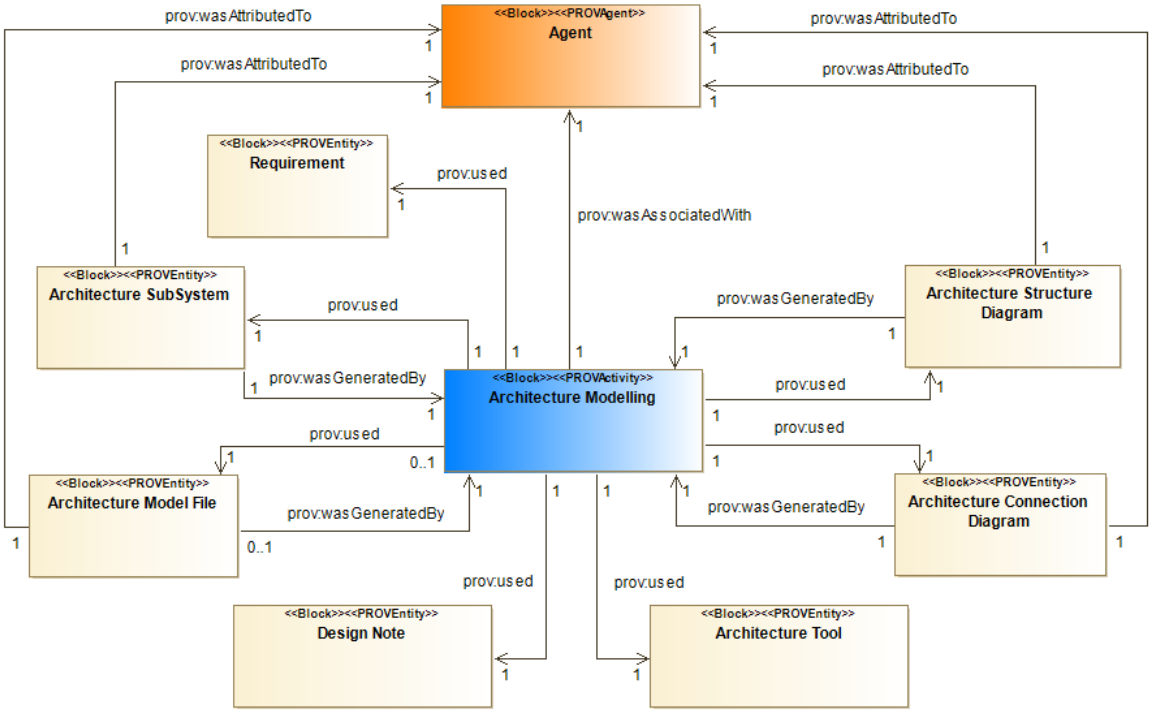


Figure 3: BDD of the *Architecture Modelling* activity

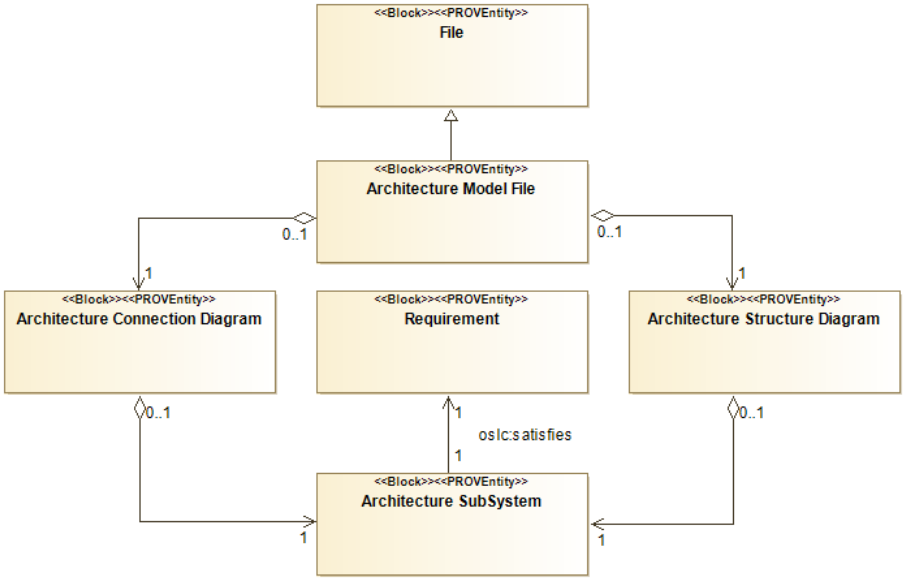


Figure 4: BDD *Architecture Modelling* elements

The activity of *Architecture Configuration Creation* is presented in Figure 5. *Architecture Configuration Creation* uses the *Architecture Model* to generate an *Architecture Configuration*.

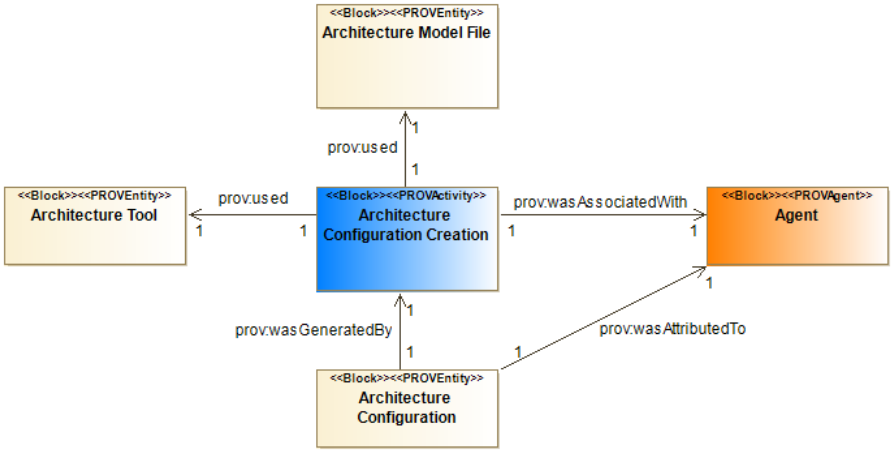


Figure 5: BDD of the *Architecture Configuration Creation* activity

A.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 6. This functionality is provided by *Modelio*.

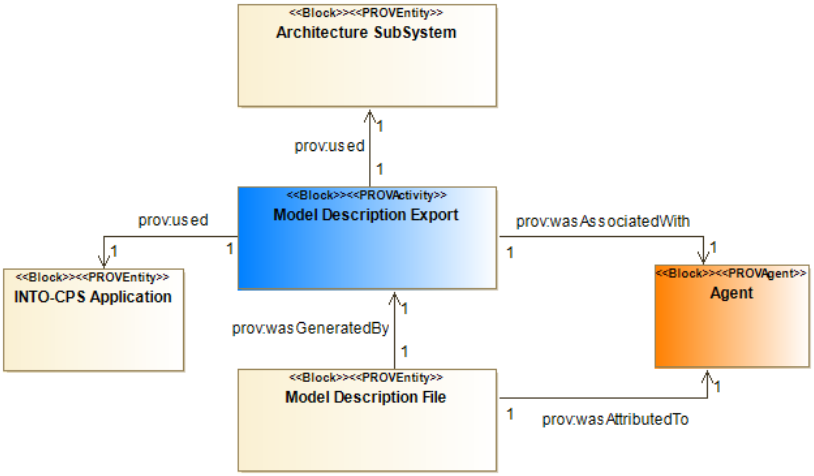


Figure 6: BDD of the *Model Description Export* activity

A.4 Simulation Models

There are two distinct activities shown in Figure 7, *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 8 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:satsifies` relation.

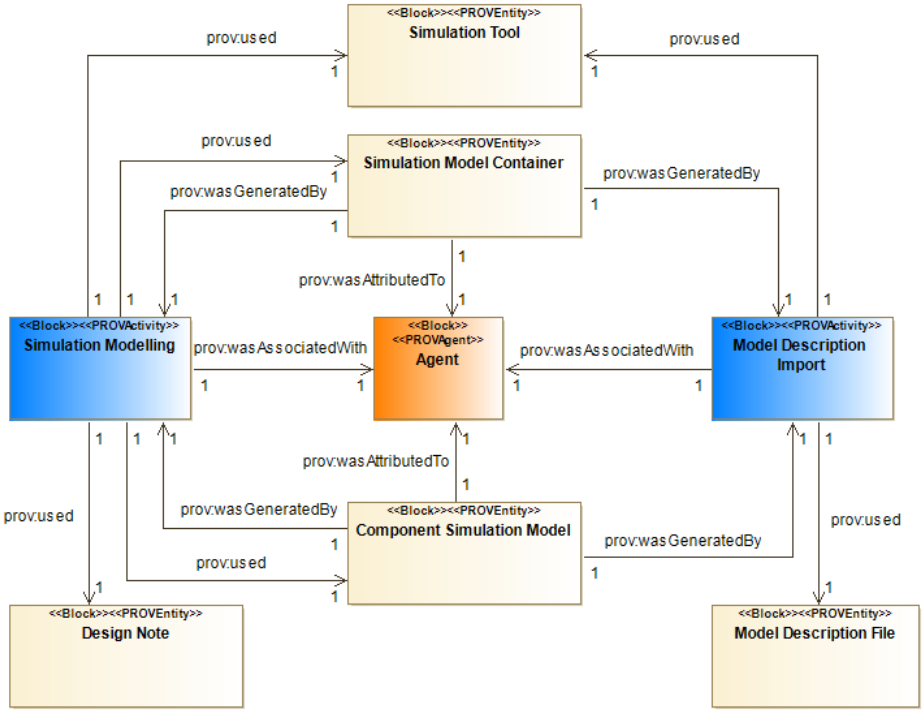


Figure 7: BDD showing *Simulation Model Creation* activities

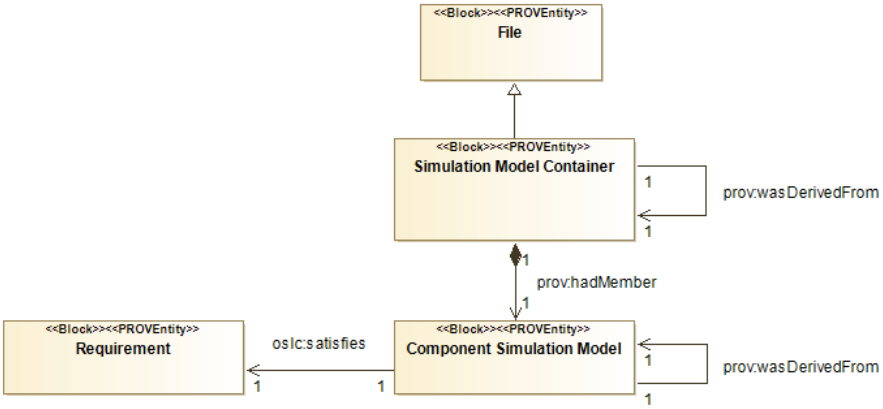


Figure 8: BDD showing *Simulation Model* files

A.5 Model Checking and Test Automation

Model Checking and *Test Automation* activities including those for the creation of models and their subsequent use for analysis. Figure 9 shows the creation of the models for model checking and Figure 10 shows their use in model checking to produce a model check result. The RT-Tester tool instantiates the abstract *Model Checking Suite* in the INTO-CPS tool chain.

Test automation also requires artefacts to perform analysis on, but in this case, there are a range of different artefacts that may be employed. These artefacts each have their own creation process, these include modelling for *Test Automation* (Figure 11), *Test FMU Creation* (Figure 12), *Mockup FMU Creation* (Figures 13 and 14). These artefacts may then be analysed by the *Test Execution Process* (Figure 15). The files associated with both model checking and test automation are shown in Figure 16.

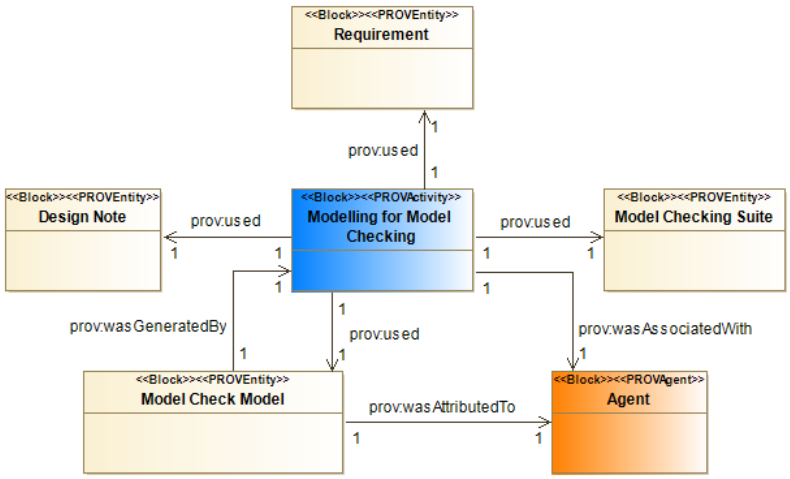


Figure 9: BDD showing the *Model Check Model Creation* activity

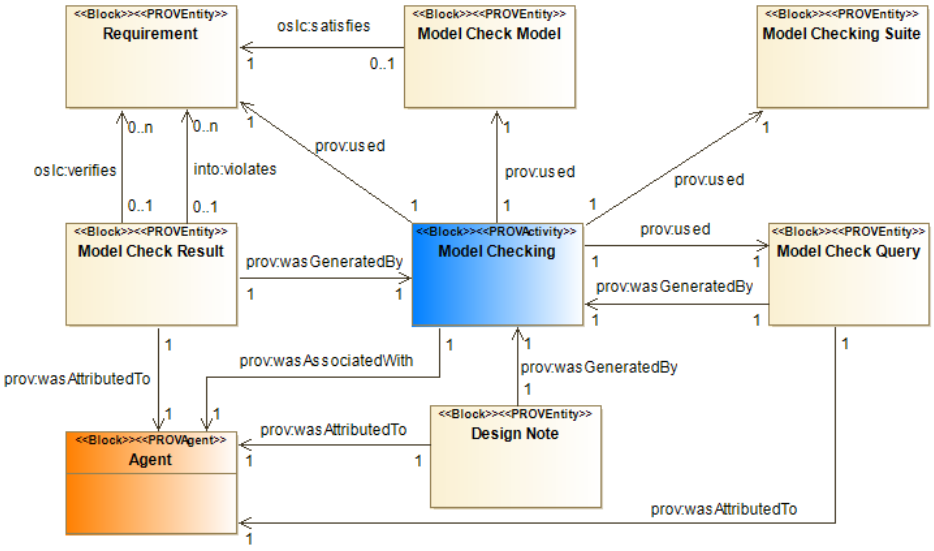


Figure 10: BDD showing the activities of *Model Checking*

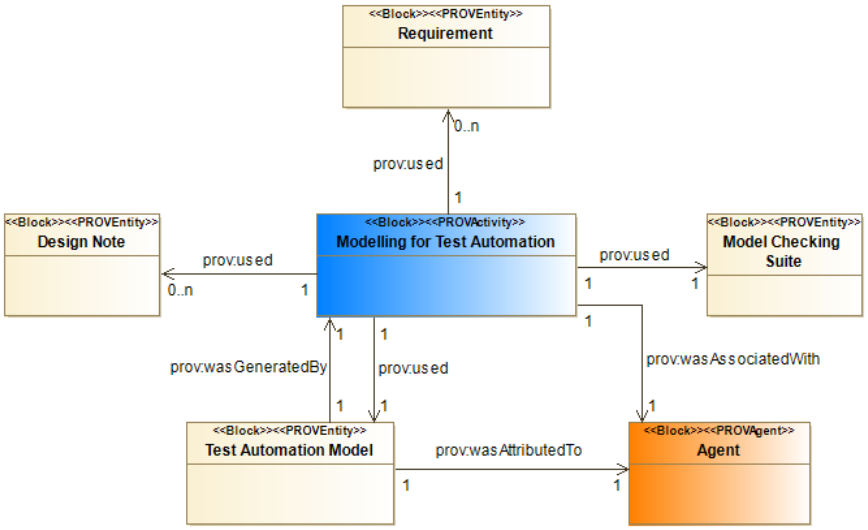


Figure 11: BDD showing the *Test Automation Model Creation* activity

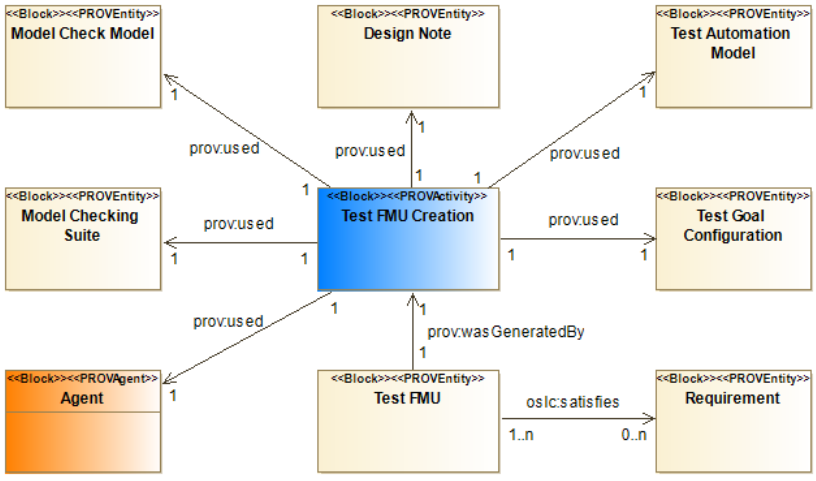


Figure 12: BDD showing the *Test FMU Creation* activity

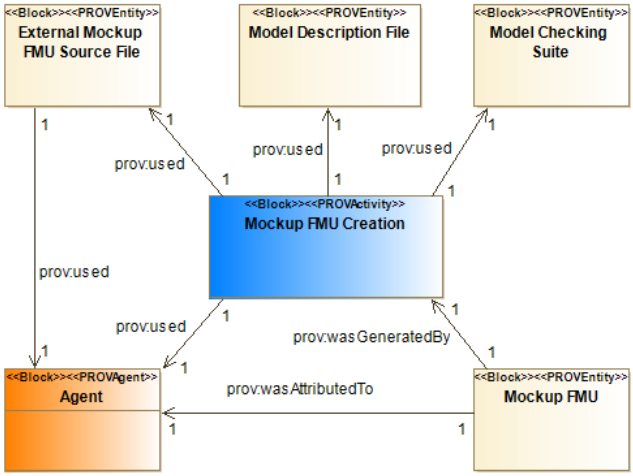


Figure 13: BDD showing the *Mockup FMU Creation* activity

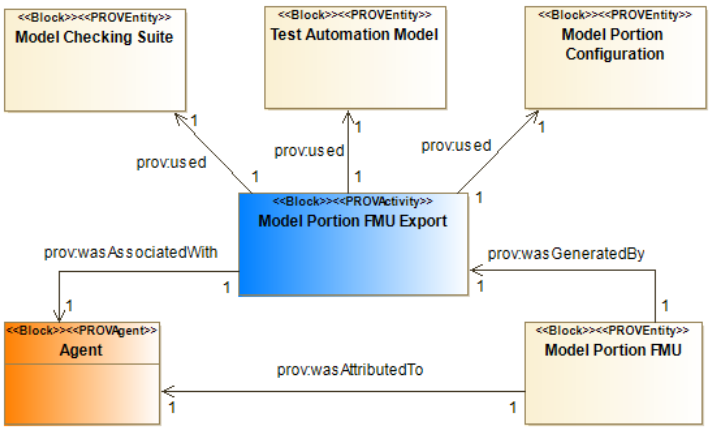


Figure 14: BDD showing the *Model Portion Export* activity

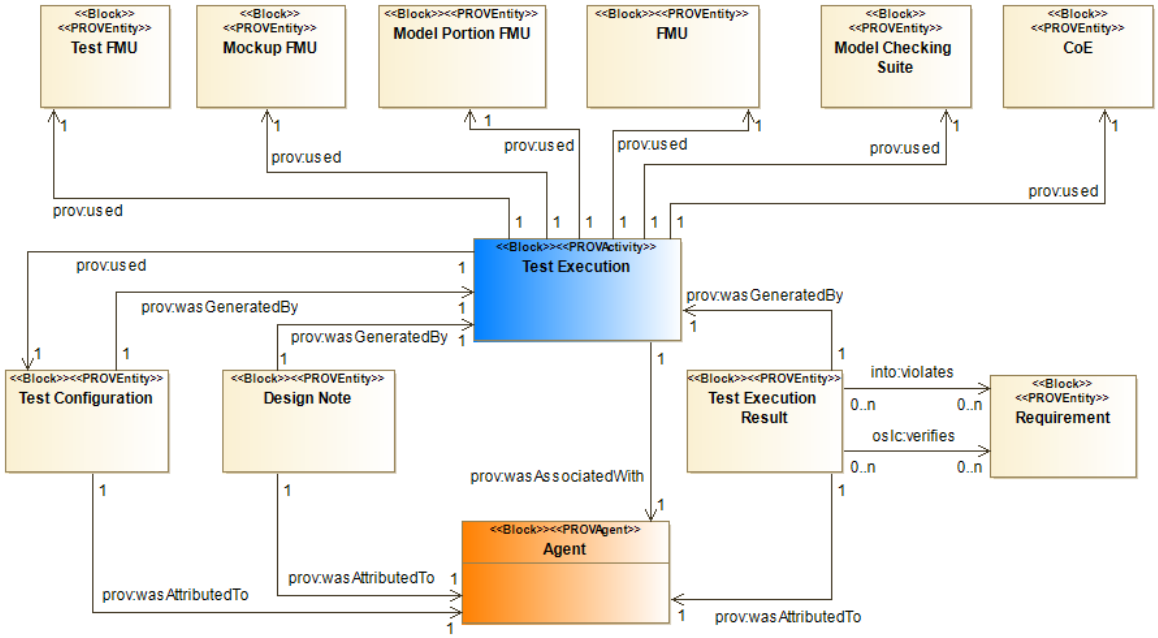


Figure 15: BDD showing the *Test Execution* activity

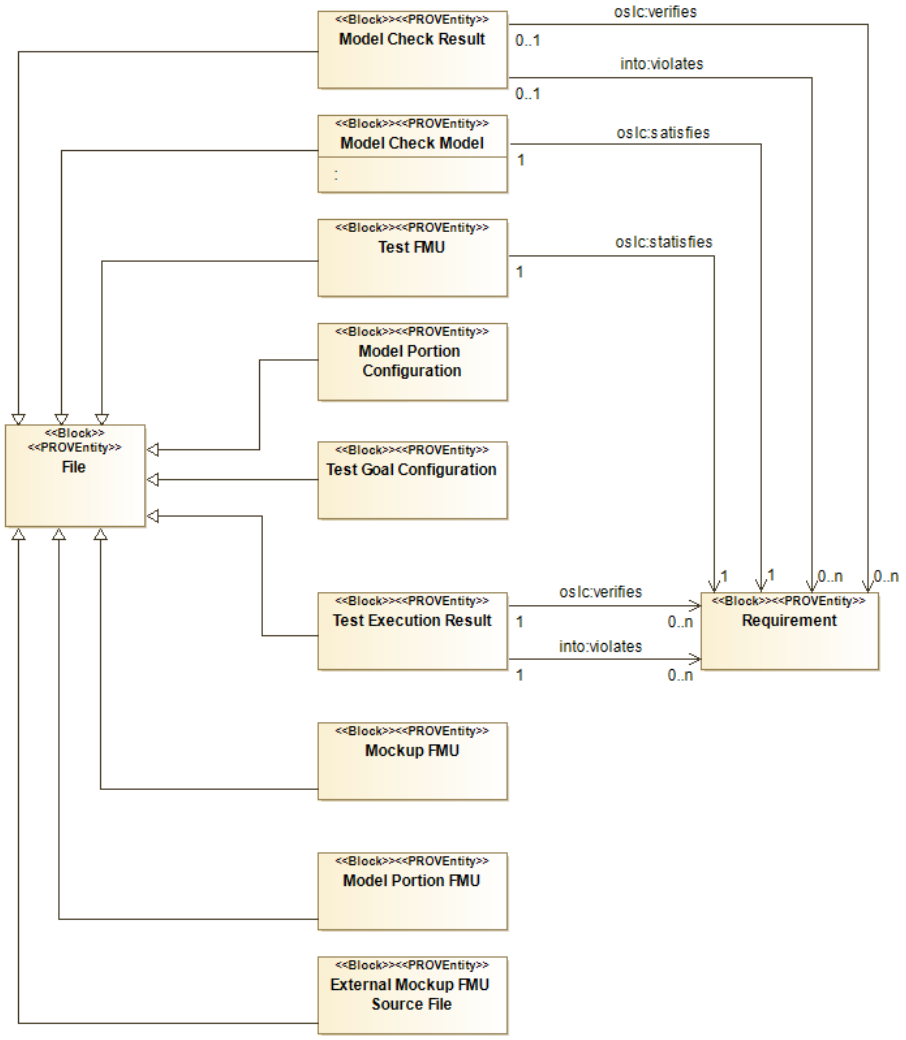


Figure 16: BDD showing the file elements used in *Model Checking*

A.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.

The creation of an FMU to support Hardware-in-the-Loop (HiL) simulation differs slightly in that instead of being generated from a *Simulation* or *Model Check Model*. It requires 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.

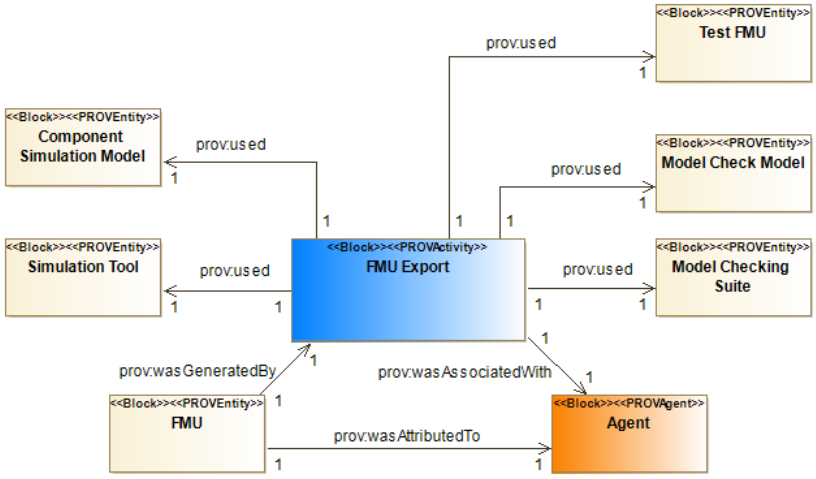


Figure 17: BDD showing the simulation *FMU Export* activity

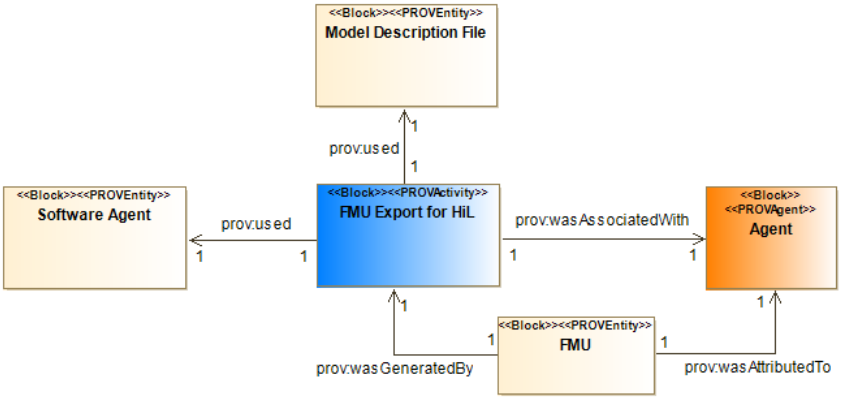


Figure 18: BDD showing *FMU Export* to allow HiL simulation

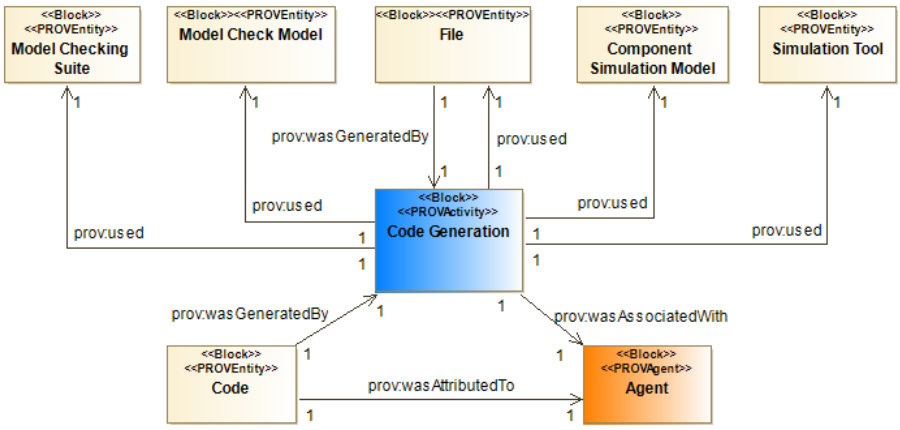


Figure 19: BDD showing the *Code Generation* activity

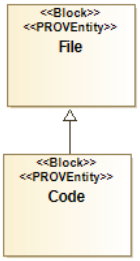


Figure 20: BDD showing the files associated with *Code Generation*

A.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*. The *Simulation Configuration Creation* activity uses the aforementioned multi-model configuration to create a simulation configuration, shown in Figure 22. 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*. Finally, Figure 24 shows the files associated with *Simulation*. The *Simulation Results* may be linked to a *Requirement*, providing evidence for or against that requirement being met.

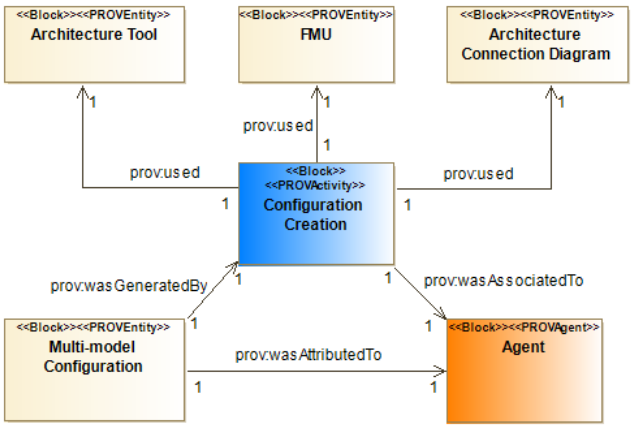


Figure 21: BDD showing the *Multi-Model Configuration* activity

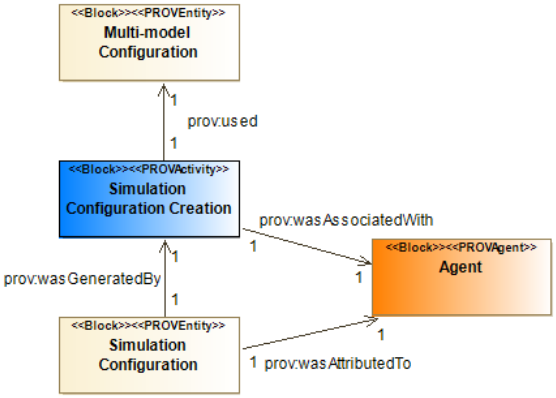


Figure 22: BDD showing the *Simulation Configuration* activity

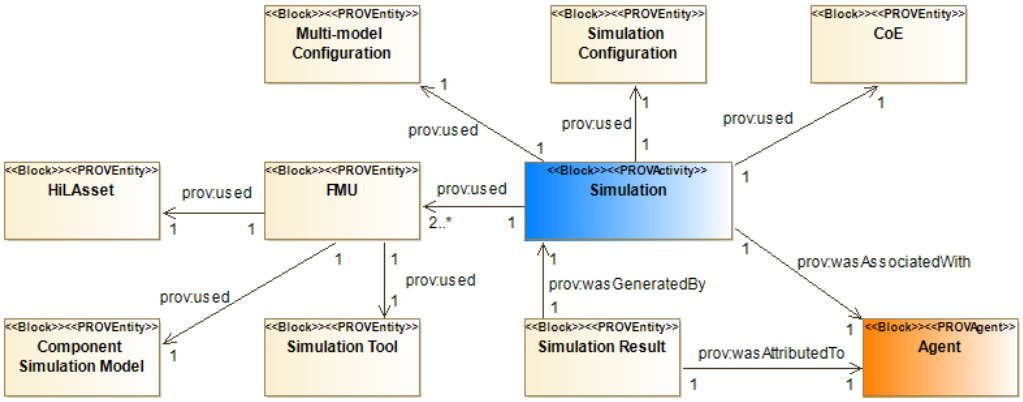


Figure 23: BDD showing the *Simulation* activity

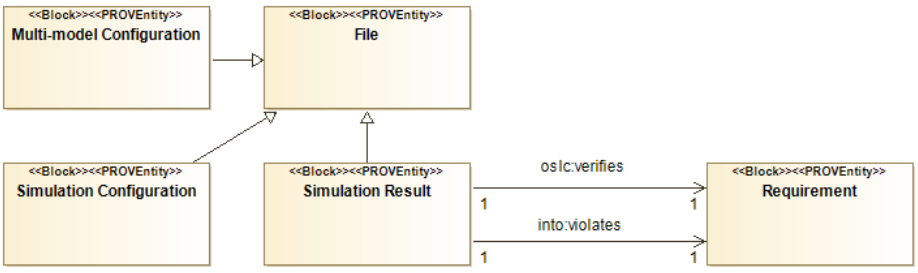


Figure 24: BDD showing the files associated with *Simulation*

A.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 26 shows the *DSE Analysis Creation* activity, which generates a *DSE Analysis Script* and *Scenario Data*. 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. 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.

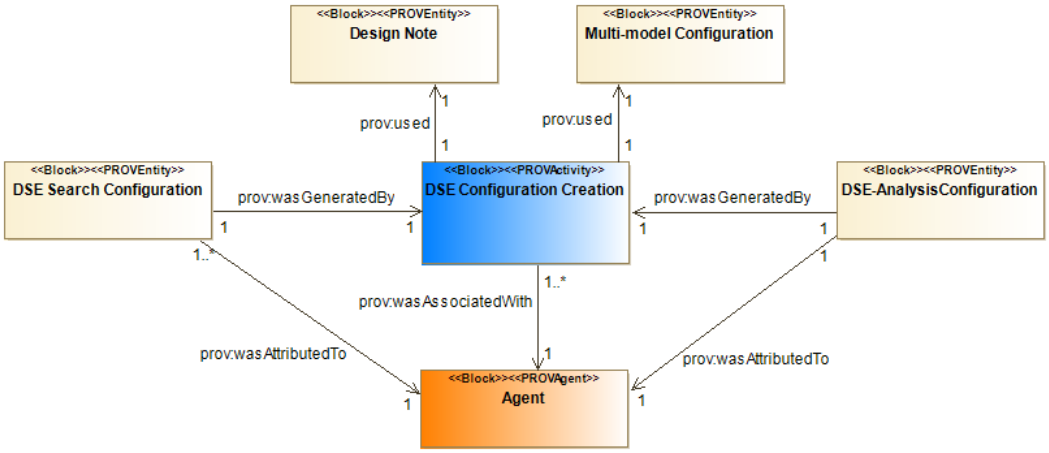


Figure 25: BDD showing the *DSE Configuration* activity

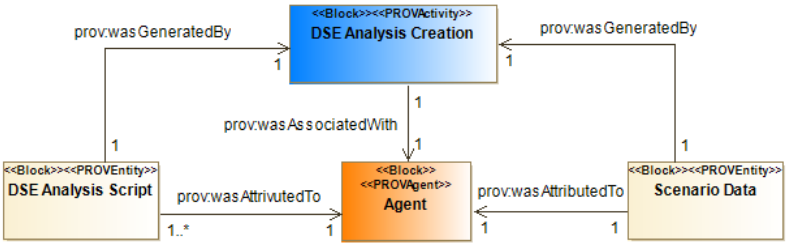


Figure 26: BDD showing the *DSE Analysis Configuration* activity

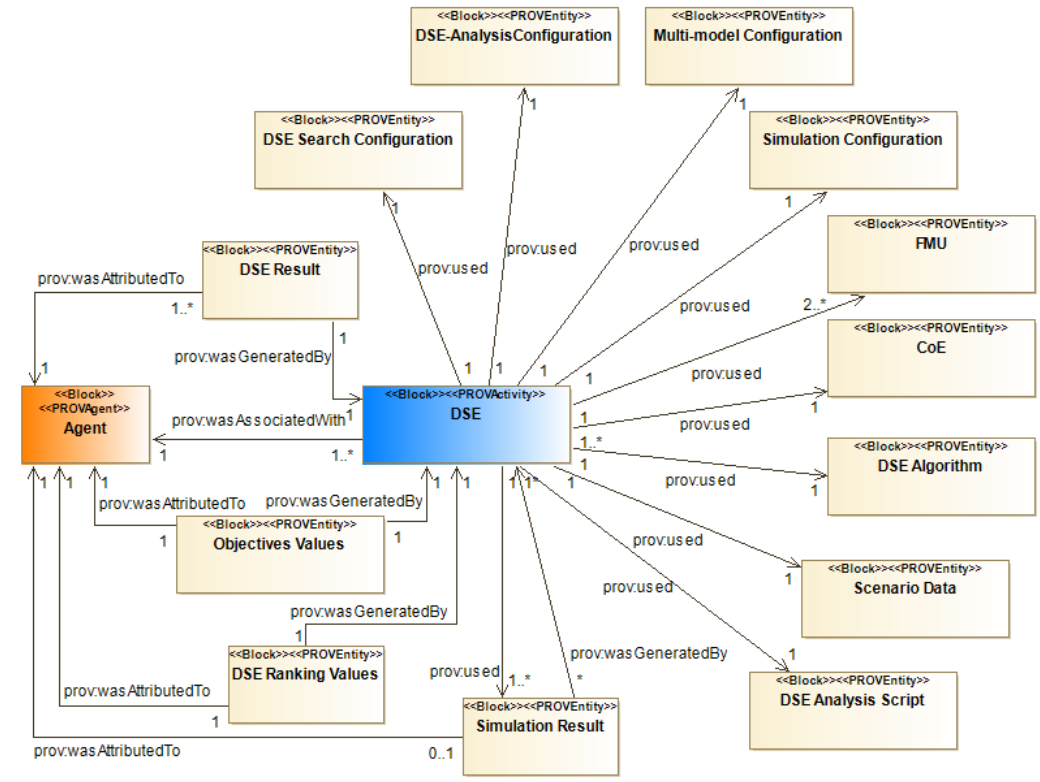


Figure 27: BDD showing the *DSE* activity

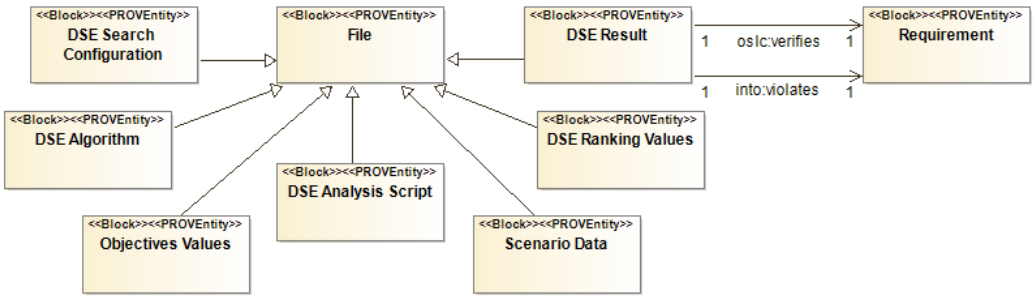


Figure 28: BDD showing the files associated with *DSE*

A.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.

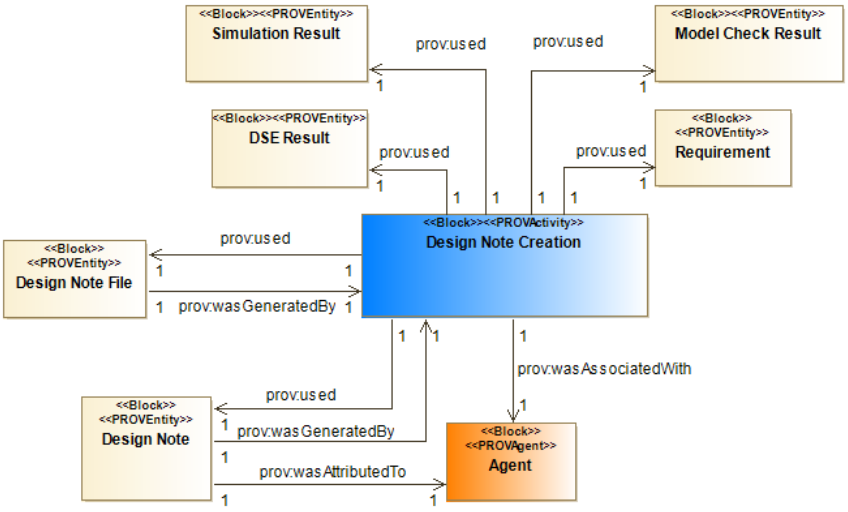


Figure 29: BDD showing the *Design Note Creation* activity

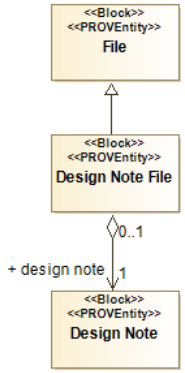


Figure 30: BDD showing the files associates with *Design Notes*

B Traceability Queries

The following are example queries in Cypher, Neo4j's query language, defined over the ontology. Some are implemented in the graphical user interface in the INTO-CPS Application (see Deliverable D4.3d [KLN+17]), while others can be executed directly in the 'expert' mode provided as well.

B.1 Impact Analysis

B.1.1 Forward traceability: requirements to entity

B.1.1.1 Find all requirements

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

B.1.1.2 Match any and all entity types

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

B.1.1.3 Match architecture entities

```
match ({uri:'Entity.r-xxx'})<-[:Trace{name:"oslc:satisfies"}]-(element)-
  [:Trace{name:"prov:wasGeneratedBy"}]-(activity)
where element.type = 'block'or
      element.type = 'architectureStructureDiagram'
return element.uri, element.hash, activity.time, element.type
order by activity.time desc
```

B.1.1.4 Match simulation models

```
match ({uri:'Entity.r-xxx'})<-[:Trace{name:"oslc:satisfies"}]-(element)-
  [:Trace{name:"prov:wasGeneratedBy"}]-(activity)
where element.type = 'componentSimulationModel' or
      element.type = 'simulationModelContainer' or
      element.type = 'fmu'
return element.uri, element.hash, activity.time, element.type
order by activity.time desc
```

B.1.1.5 Match simulation results

```
match ({uri:'Entity.r-xxx'})<-[:Trace{name:"oslc:verifies"}]-(element)-
  [:Trace{name:"prov:wasGeneratedBy"}]-(activity)
where element.type = 'simulationResult'
return element.uri, element.hash, activity.time, element.type
order by activity.time desc
```

B.1.2 Backwards traceability: FMU to requirements

B.1.2.1 Find all FMUs

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

B.1.2.2 Match requirements that affect an FMU

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

B.1.3 Backwards traceability: simulation model to requirements

B.1.3.1 Find all FMUs

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

B.1.3.2 Match requirements that affect an FMU

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

B.2 Simulation Sources

B.2.1 Find all simulations

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

B.2.2 Find Sources and Sinks for a Simulation

B.2.2.1 Find sources

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

B.2.2.2 Find sinks

```
match({uri:'Entity.simulationResults-xxx'})-
  [:Trace{name:"prov:wasGeneratedBy"}]->(simulation)<-
    [:Trace{name:"prov:wasGeneratedBy"}]-(entity)
return entity.uri, entity.path, entity.hash
```

B.3 Coverage

B.3.1 Requirements without architecture elements

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

B.3.1.1 Requirements without simulation models

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

B.3.1.2 Requirements without positive simulation or test results

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

B.3.1.3 Requirements without any simulation or test results

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

B.3.1.4 Requirements with at least one positive but no negative test results

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

B.4 User Impact

B.4.1 Find all known users

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

B.4.2 All artefacts influenced by a user

B.4.2.1 By user URI

```
match (usr{uri:'Agent.xxx'})<-[:Trace{name:'prov:wasAttributedTo'}]-(entity)
return entity.uri, entity.type
```

B.4.2.2 By username

```
match (usr{name:'xxx'})<-
  [:Trace{name:'prov:wasAttributedTo'}]-(entity)
return entity.uri, entity.type
```

B.4.3 All activities performed by a user

B.4.3.1 By user URI

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

B.4.3.2 By username

```
match (usr{name:'xxx'})<-[:Trace{name:'prov:wasAssociatedWith'}]-(entity)
return entity.uri, entity.type
```