 THE RESEARCH LEADING TO THESE RESULTS HAS RECEIVED FUNDING FROM THE EUROPEAN UNION'S SEVENTH FRAMEWORK PROGRAM (FP7/2007-2013) FOR CRYSTAL – CRITICAL SYSTEM ENGINEERING ACCELERATION JOINT UNDERTAKING UNDER GRANT AGREEMENT N° 332830 AND FROM SPECIFIC NATIONAL PROGRAMS AND / OR FUNDING AUTHORITIES.

CRitical S**YST**em Engineering **A**cce**L**eration

Specification, Development and Assessment for Simulation Models – V1

D613.011

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	Specification, Development and Assessment for Simulation Models – V1
Deliverable No.	D613.011 (merged with D613.021)
Dissemination Level	CO
Nature	R
Document Version	V1.1
Date	2014-03-13
Contact	Gerald Stieglbauer
Organization	AVL
Phone	+43 316 787 7803
E-Mail	gerald.stieglbauer@avl.com

AUTHORS TABLE

Name	Company	E-Mail
Gerald Stieglbauer	AVL	gerald.stieglbauer@avl.com
Selver Softic	VIF	selver.softic@v2v2.at
Cecilia Ekelin	VOLVO	cecilia.ekelin@volvo.com
Alberto Melzi	CRF	alberto.melzi@crf.it
Aleksander Lodwich	ITKE	aleksander.lodowich@itk-engineering.de

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
0.1	21.12.2013	Initial document	all
0.2	13.01.2014	Document ready for internal review	
0.3	28.01.2014	Document ready for external review	
1.0	10.03.2014	Final document	all
1.1	12.03.2013	Merge this document with D_613_021_v1-0	all

CONTENT

Content of Figures

Content of Table

1 Introduction

Simulation models still have an increasing impact within a huge variety of development processes. In typical Hardware- or Software-in-the-Loop (HiL or SiL environments, for instance, simulation models replace real-world objects in order to allow rapid prototyping or test frontloading. Powerfulness simulation tools such as MathWorks Simulink lead to great flexibility regarding the operation purpose of simulation models. As a consequence, simulation models can be used in very early as well as in very late development process stages. This, however, leads to the problem that some demanding characteristics and constraints of simulation models (such as simulation accuracy, real-time constraints, etc.) differs significantly in the various development stages and thus often hinder model re-use and model development collaboration. Besides this, a lack of model development collaboration activities is still often found between different projects. Even if participants of the projects are aware of each other, there is often no straightforward access to the applied simulation model in order to analyse them regarding their potential of re-use.

1.1 Role of deliverable

The major objective of this work package is therefore to significantly improve collaboration and re-use of simulation models or, where constraints such as mentioned above hinder the development and use of consolidated simulation models, setting models with similar purposes in corresponding relation to each other. In addition, the simulation models should be more straightforward accessible and findable in terms of their purpose to significantly improve project and model development collaboration. This includes especially the possibility to apply requirement and variability management and is thus be related to the work packages 6.7 (Requirement Based Engineering) and 6.10 (Variability Management) of SP6. Finally, an improved degree of automation and a reduced set of development overhead should be other key results of this work package. Of course, all collaboration aspects which include other tools and/or technological bricks have to fully comply with the interoperability specification defined in work package 6.1.
In this deliverable, Bricks are documented which are developed in the WP6.13 (Simulation Models). This deliverable is updated iteratively, i.e. three times during the project runtime, based on the corresponding milestones of the project. Therefore the Brick documentations in this document represent an evolutionary process of continuous development and enhancement of the CRYSTAL solutions, and complementary to previous version of this deliverable.
In the current iteration round the following improvements are implemented by corresponding technical items (Tis) as described in more detail by the following chapters:

- AVL Santorin functionality extension to act as a simulation model data backbone
- AVL Navigator functionality extension for the simulation model data backbone aspects
- Simulation model data exchange
- Definition of an OSLC data model for simulation model dependent tool interoperability
- Tracing and linking infrastructure functional block to connect Simulink block or state-flow model to an architectural component
- Adapter for exchange in workflow between Simulink and Enterprise Architect with involvement of PTC Integrity
- Requirement Traceability to Simulink Models
- Adapter for tracing, creation, maintenance and linking of requirements to architectural components and Simulink
- Simulation model exchange across development phases and working groups
- Tracing and exchange data backbone between Enterprise Architect and Simulink
- Co-Simulation
- Support for different kind of simulation models for AVL ARTE.Lab
- Establishing simulation model relations with OSLC in a co-simulation environment

1.2 Relationship to other CRYSTAL Documents

This document is related to several WPs in SP6 (WP6.3, 6.7, 6.8, 6.10, 6.12) and to some use case work package, especially to WP3.4 in the automotive domain. For detailed information to related tasks see the description header of the several tasks of WP6.13.

1.3 Structure of this document

The structure adheres to the template suggested by the SP6 coordination team. Each brick assigned to WP6.13 is related to a task and may be related to one or more technical items depending on the related use case needs. If the use case needs are not evaluated yet at this stage of the CRYSTAL project, this will be mentioned accordingly and a description will be delivered at a later stage of the project.
Due to the suggestion of the interim review Meeting in Brussels on the 11th of February to reduce the Deliverables to one per WP per report period, this version of the document is merged with the deliverable D_613_021_v1-0 (Development of the simulation model data backbone as described in T6.13.1 in Chapter 2). This deliverable describes prototype implementation activities for the simulation model data backbone. A separate Chapter 6 is introduced in addition to the SP6 template structure.

2 T6.13.1: Simulation Model Data Backbone

2.1 Description

Name:	Simulation Model Data Backbone
Contact:	gerald.stieglbauer@avl.com
Dependencies	WP3.4, T6.3.15, T6.10.7, T6.13.2, T6.13.3, T6.13.4, T6.8.9
License	
Additional information	

Simulation models play a central role in the field of vehicle testing. Independent on the development stage of a vehicle, simulation models are executed to simulate the testing environment (street, driver, weather condition, etc.) or parts of the vehicle, which are not implemented yet by physical components (rest vehicle simulation). In case of development frontloading with the use of simulation models, even the entire vehicle may be constructed in form of a simulation models. Important test procedures such as vehicle calibration iterations can thus be performed independently from the development stage of a vehicle.
Various tools around this simulation task are used to create different kinds of artefacts throughout the testing process (e.g. model configurations are changed constantly during calibration tasks). Most of them are currently stored locally or in a non-transparent manner (and thus are hardly to access), which makes data-reuse and traceability complicated or even impossible. This becomes even truer due to the fact that different tool sets may be used per development phase. Each tool set is assigned to a separate test case development process, represented by its own testing V-model.
The *AVL Simulation Model Data Backbone* (called just *AVL Data Backbone* in the following) is a generic concept to overcome these limitations. Figure 2-1 below illustrates the basic idea of this concept as applied in WP3.4.

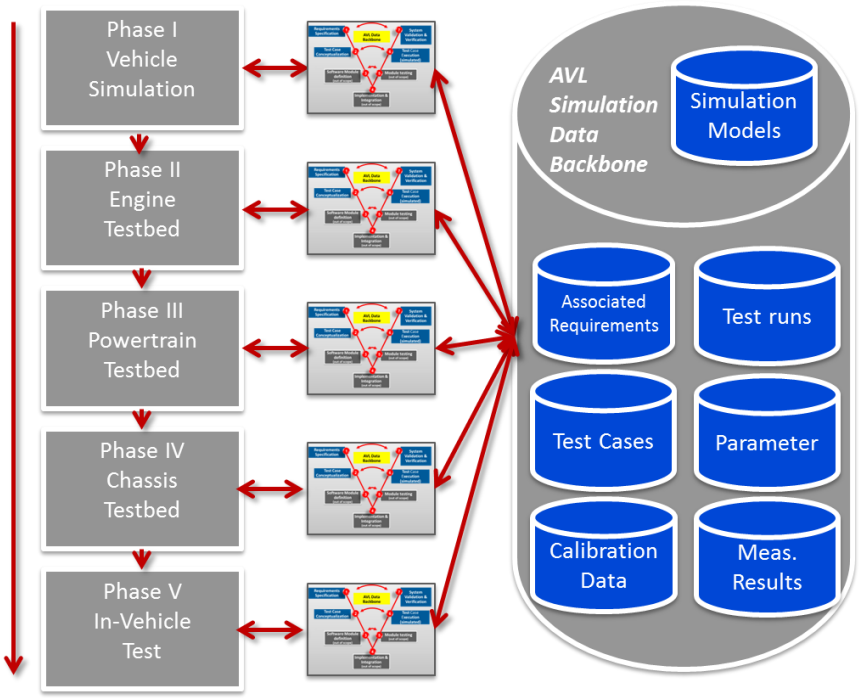

Figure 2-1 The AVL Data Backbone concept

The AVL Data Backbone a kind of single-source-of-truth for all tools and data categories related to simulation models and applied in all testing phases represented by different testing V-models (and thus different tools). With this concept simulation model consistency among the development processes should be enabled and effective frontloading of development tasks becomes possible.

2.1.1 Generic AVL Data Backbone Scenario

The following scenario description (illustrated by Figure 2-2) details the major purpose and objective of the AVL Data Backbone:

1. Within a certain project and at a particular development stage, simulation models X are created for a specific purpose. These models describe several aspects of a certain object of interest (e.g. a mechanical part, the behaviour of a certain device, complex physical processes, etc.). The models are developed and embedded in certain development and simulation environments, whereas each environment may have its own execution semantics.
2. At different development stages of the same project or in an arbitrary development stage of another project, other simulation models Y are created with a similar or related purpose than simulation models X (see picture below). Due to the constraints, boundaries or requirements of the project or the development stage, the modelled aspects of Y may vary more or less significantly from the simulation model X, however, may also be comprised of overlapping or related aspects. Besides the deviation of the purpose of model X and Y, constraints such as simulation granularity and real-time requirements determine the model's content and simulation platform. Especially in this case, however, models X and Y may also be comprised of overlapping or related aspects.

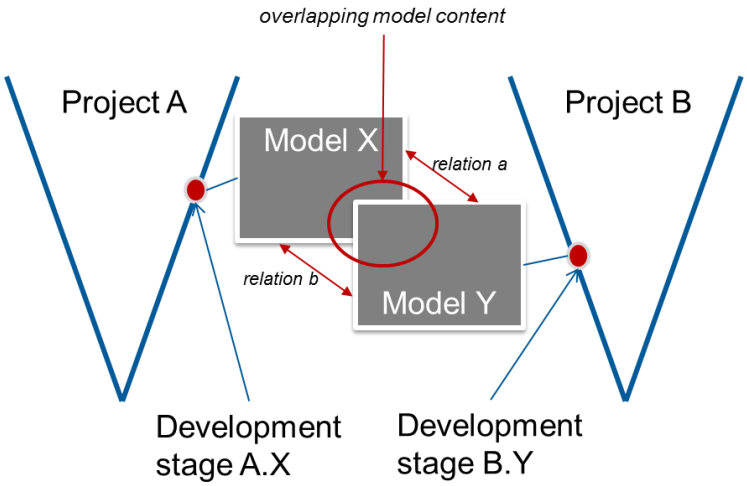

Figure 2-2 Interrelation between two models in two different development processes

3. As a consequence of 1) and 2) the following limitations would occur:
   a. Various model aspects (including the parameters and common variants of the modelled object) have to be re-modelled in every different project (or even development stage)
   b. Due to the lack of model sharing, the participants of the projects (or even the participants in the same project at different development stages) have limited or no access (and have thus as well often no awareness) of the model in other projects (or development stages).
   c. Even if the awareness is given, it is still a standard situation that interrelated data has to be exchanged manually with all the known limitations (e.g. lack of consistency, error-prone step, time efforts, etc.)
4. The CRYSTAL project will provide a significant contribution to the described limitation by providing an interoperability standard, which should provide a more straightforward access to the simulation platforms and its simulation models.
5. In addition to interoperability aspects, a centralized simulation model management environment (e.g. a simulation model data backbone) enables further perspectives to improve model development collaboration including the definition of relations between models.

6. Consequences of 4) and 5) are thus:
    a. Project collaboration becomes more straightforward (e.g. introducing a simulation model development categories and model development status across projects or development stages)
    b. An improved degree of automation (e.g. automatic synchronization of model data)
    c. A reduced set of development overhead (e.g. re-using of models)
    d. A standardized way how to access modelling data (e.g. introducing model categories, complying with interoperability standard)
    e. Enables the possibility of defining, exchanging and aligning of commonalities between models (e.g. model parameters)
    f. Should act as a door opener for further aspects such as versioning and variant management, applying requirement management, etc.

In the following, two examples that would benefit from this generic scenario are described

Example 1) WP3.4 consists of two sub-use cases. UC3.4a is about interoperability in the context of vehicle testing and calibration iteration based on model-based requirement management. UC3.4b is about development and testing of ECU SW. In both scenarios simulation models heavily used. In UC3.4a, simulation models are used to enable vehicle testing frontloading in early vehicle development steps or to perform rest vehicle simulations, whereas only some vehicle components (i.e. unit under test) are tested physically on a corresponding test bed (e.g. engine testbed). Consequently, simulation models are applied in different development stages.

Example 2) Such simulation models, however, are not limited only in the context of testing a certain physical vehicle part (e.g. an engine). In UC3.4b, the same simulation models are re-used during ECU control software development. In this case, simulation models are generating input values for the ECU control software and/or the output values of the ECU control software is used by the simulation models in order to evaluated the expected behaviour. Again, simulation models (and all their related data) are re-used in total different development stages with different project teams.

In the current state of the project further model re-use questions such as re-using only parts of a model or creating variants of a model are left open. These issues may be addressed at a later stage of the projects when first prototypes are implemented and evaluated.

2.2 Use Case coverage and application

2.2.1 Coverage in WP3.4

In WP3.4, an interoperability approach based on IOS should be evaluated for the AVL data backbone concept. Especially the concepts of one IOS candidate, namely OSLC, should be considered here. Figure 2-3 illustrates a possible architecture based on the OSLC concepts. The various data categories are stored in one or even more (3rd party) data providers. The data consists of all the details created by the related authoring tool and only these authoring tools are able to fully interpret and modify this kind of data. OSLC adapters, however, abstract from these details and provide only a reduced data model per data category (also called OSLC domains). These (potentially standardized) OSLC domains are designed in a minimal manner in order to just fulfil the needs for defining data interrelations and navigation across the data categories. On top of this minimal OSLC data structure a *uniform workbench* could navigate over this data structure, without the need of understanding all the details the authoring tools have to deal with. If a deeper data analysis or modification is needed, the workbench just delegates this task by invoking the corresponding tool with the appropriate OSLC link or requests an appropriate data artefact representation.

In addition, an AVL customer has the freedom of choice which kind of data backbone he wants to use. For instance, a classical ALM tool such as PTC Integrity may provide important features such as variant and version management. However, this feature may not be needed by every customer, consequently another data provider (e.g. an AVL data backbone) is sufficient. With the use of OSLC both the authoring tools as well as the uniform workbench does not depend on which data provider is in use.

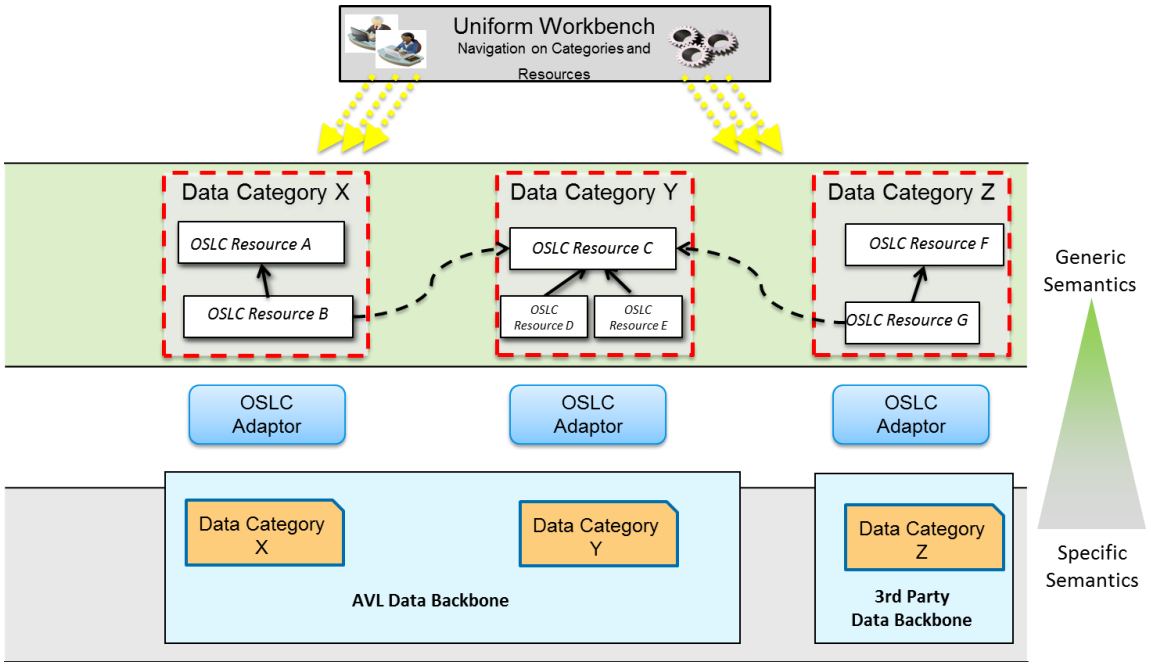

Figure 2-3 Integration of OSLC and the AVL Data Backbone concept

Furthermore, with such a concept data consistency in the area of simulation models and their depending data categories have to be ensured. It is important to understand what consistency means in this context. For some situations it is maybe sufficient that a central data repository ensures the single-source-of-truth concept, i.e. a unique storage location and transparent way of data access for the involved applications. In practise, however, a single-source-of-truth concept does not necessarily causes consistency of data content (e.g. re-using the same set of calibration data throughout two testing phases). It may happen that for various reasons (e.g. if different naming conventions are common in different development phases) two variants of calibration data sets are created at two testing phases.

A stronger meaning of consistency may forces that the data content of the several data categories is aligned with the different testing phases. For instance, with a fully consistent data set, the parameters, calibration data, and measurement results of two test-runs in different test phases phase became directly re-useable and comparable. It is part of the project to evaluate, which interpretation of consistency is sufficient to overcome the most important limitations of today's systems.

2.3 General Improvement

2.3.1 Improvements for WP3.4

TI NAME: AVL Santorin functionality extension to act as a simulation model data backbone					
TI_ID	CRYSTAL_TI_6.13.1_1	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com

Description:

In the context of the use case scenarios in WP3.4, the data backbone plays a central role. Starting from the (formalized) requirements, all the data artefacts created in the following testing phases (especially simulation models) should be stored and managed by such a data backbone concept. The selected tool of choice for such a data backbone implementation will be *AVL Santorin*. AVL Santorin is traditionally used for managing measurement result data and complies with the ASAM ODS standard. On the other hand, it is also a database that is able to hold other data as well. In terms of version and variant management, however, the possibility is limited and alternative/parallel solutions may be considered for the use case scenario (e.g. by integration of a PLM tool).

Consequently, the functionality of AVL Santorin has to be enhanced in order to fulfil these requirements. A key issue here will be the integration of other data sets than measurement results targeting especially simulation models including their configuration data. This includes also the

relations of simulation models and the testing environment (e.g. test bed configuration) in the context of rest vehicle simulation.
Link to internal working documents: **See Deliverable D_613_012**

TI NAME: AVL Navigator functionality extension for the simulation model data backbone aspects					
TI_ID	CRYSTAL_TI_6.13.1_2	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com
Description: In order to enable a global data management of simulation models and their belonging data, a navigation tool should be developed or extended with the ability to navigate on a high-level data structure of the various data categories. These concepts, for instance, would include that this navigator should provide the possibility to delegate functionality of editing and interpreting the specific data content to specialized tools.					
Link to internal working documents:					

2.4 Integration and Interoperability

2.4.1 Improvements for WP3.4

TI NAME: Simulation model data exchange					
TI_ID	CRYSTAL_TI_6.13.1_3	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com

Description:

Based on the implementations of the technical items CRYSTAL_TI_6.13.1.1 and CRYSTAL_TI_6.13.1.2, these implementations should be applied to demonstrate a fully integrated requirements engineering process including requirement verification and data migration over development phases by applying the CRYSTAL IOS concepts. OSLC as one candidate of a major technology in IOS should be evaluated here. In terms of OSLC, is mandatory to develop a corresponding OSLC resource model that interlinks the data categories appropriately.
Especially for simulation models, two sub use cases defined in WP3.4 should be combined via the data backbone concept based on OSLC. In Figure 2-4, these two V-model variants are sketched. The first variant (UC3.4a) has a strong focus about test case compilation and verification based on a set of requirements defined by a corresponding requirement model. The second variant (UC3.4b) deals especially with the development of the embedded verification platform to fulfil the requirement during the development. Both UC variants will be based on the use of simulation models for their corresponding purposes.
More than just being two separate instances of the common base UC, both variants should be combined as well in the following way: Since both use case variants are heavily based on simulation models, whereas the same, similar or related models are applied in both use cases (or are at least interrelated by certain simulation model parameters), the implementation of an appropriate Simulation Model Data Backbone is essential to share certain models between the use cases.

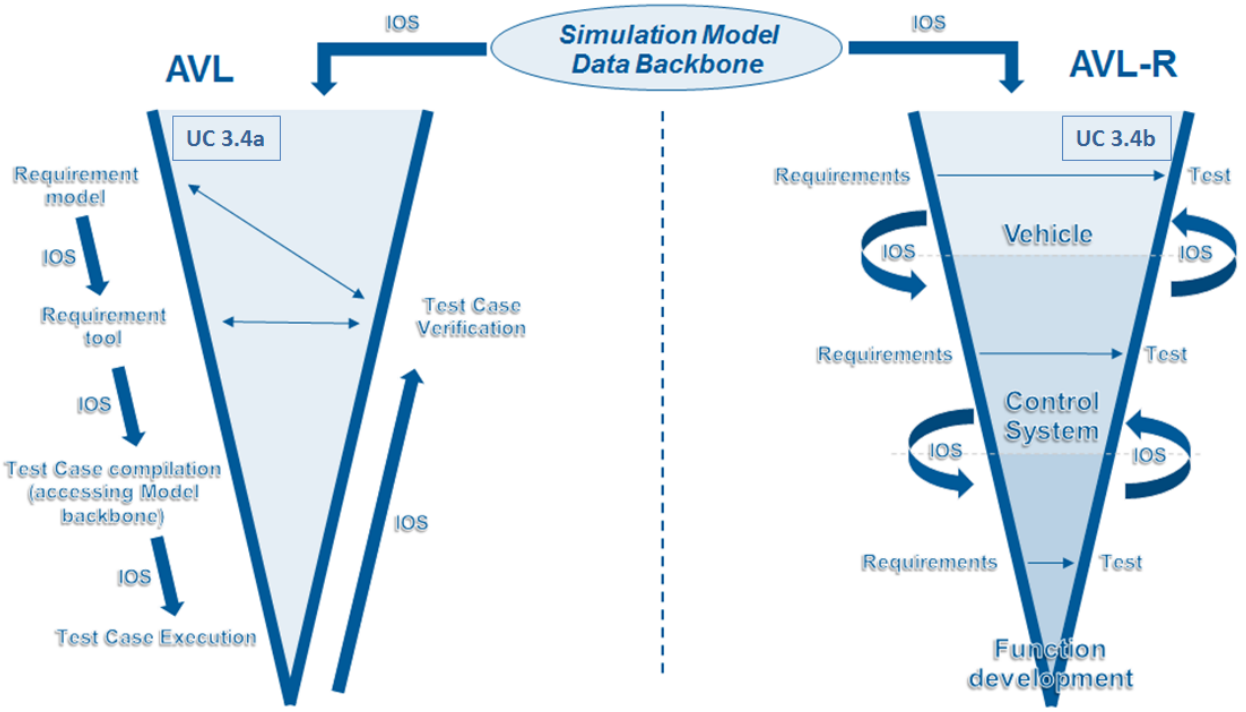

Figure 2-4: Model exchange between two different development process via IOS interfaces.

In case of UC3.4a for instance, simulation models for certain control systems (such as an engine control unit) are needed. These simulation models are provided by UC3.4b usually in form of appropriate Simulink models. As described as an engineering method in WP3.4, these models may are used for co-simulation in an early testing phase in order to simulate the behaviour of an ECU in a particular vehicle, whose physical behaviour including its environment is (partly) simulated as well (e.g. for calibration iterations).

On the other hand, the control model needs to be tested as well, which is accordingly described by the V-model of UC3.4b. In order to test such functions without have the possibility to do that directly in a ready-to-use vehicle, vehicle simulations are needed. The AVL simulation tools Cruise and Boost are specialized tools for such kind of simulation models. These tools are applied at UC3.4a and are able to provide appropriate models for UC3.4b.

In Figure 2-5, a general pattern with the use of OSLC on top of various data backbones is presented. Let's assume that there is an in-house data backbone (called in AVL Data Backbone in case of WP3.4a), which stores various data categories assigned to the testing V-models of UC3.4a. In addition, however, further data backbones have to be considered, e.g. provided by third party tools (such as PTC Integrity in case of UC3.4b). Depending on the data categories stored in these data backbones, a meta-model structure has to be defined on top of the data backbones that abstracts from the detailed content of each data backbone. This means in other words that these meta-model is defined independent of the location of storage on the one hand and is limited to only those data entities (across the data categories) that need to be linked to each other.

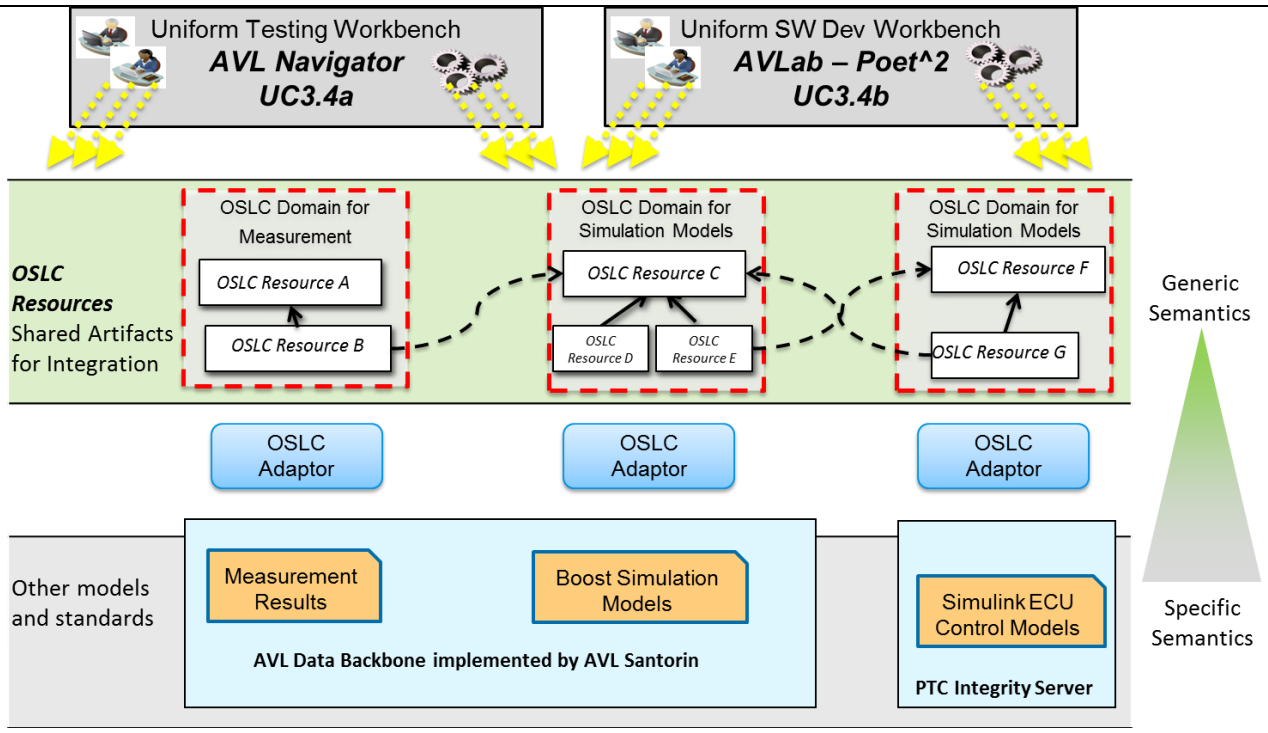

Figure 2-5 Possible OSLC architecture for exchanging simulation models

A possible implementation of the AVL Data Backbone is by using the AVL tool Santorin. Santorin is originally specialized on the management of measurement results of testing cycles and adheres to the measurement-related ASAM ODS standard. Since this tool is based on a database, it can be also used for the storage of other data entities (for the given example this would be for instance Boost simulation models needed by UC3.4b). In addition, the ASAM ODS provided possibilities of extending its data storage model if needed. In addition, Santorin comes up with a Navigator frontend to give support for browsing on the data entities stored in the database.

By introducing an OSLC architecture such as described before, the Navigator part of Santorin could now browse on a certain instance of the OSLC resource model. It could implement its own business model that reflects how to relate the different data set stored in the Santorin database. If details of the data are needed to be shown or edited, the Navigator just opens the adequate tool, which would be the built-in functionality in case measurement results and Boost in case of simulation models.

In addition, however, the Navigator would be also capable to browse on other data sources as Santorin. In UC3.4b PTC Integrity is used for data storage, which also holds the corresponding Simulink models for the ECU control software developed by this use case. The OSLC data structure thus enables establishing of relations across data backbones and development processes: For a specific project in UC3.4 a direct link to an ECU simulation model of UC3.4b could be established.

Correspondingly, from the UC3.4b point of view, the same mechanism can be applied: Needed vehicle simulation models (e.g. Boost models) developed in UC3.4a are referenced via corresponding OSLC links. The tool AVLab, used in UC3.4b, operates on the OSLC data structure (it has not to be the same instance as in UC3.4a) as well.

Link to internal working documents:	

TI NAME: Definition of an OSLC data model for simulation model dependent tool interoperability					
TI_ID	CRYSTAL_TI_6.13.1_4	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com
Description: Test and calibration iterations based in simulation models play a central role in WP3.4. Based on the proper					

definition of requirements test and calibration iterations are performed as illustrated in Figure 2-6. The pattern consists of five parts called requirement definition, calibration set-up, test bed and UUT set-up test-run simulation/execution and iteration results.

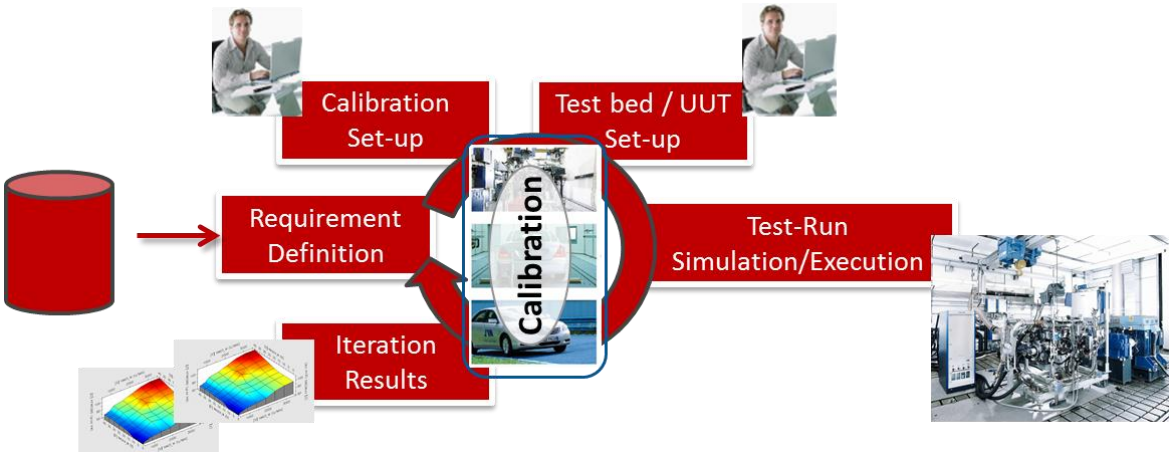

Figure 2-6: The test and calibration pattern

As illustrated in Figure 2-7, four of these five parts (except the test-run simulation/execution; see middle part of the figure) can be naturally associated to five data categories (upper part of the figure) as well to concrete data artefacts as shown in (lower part of the figure).

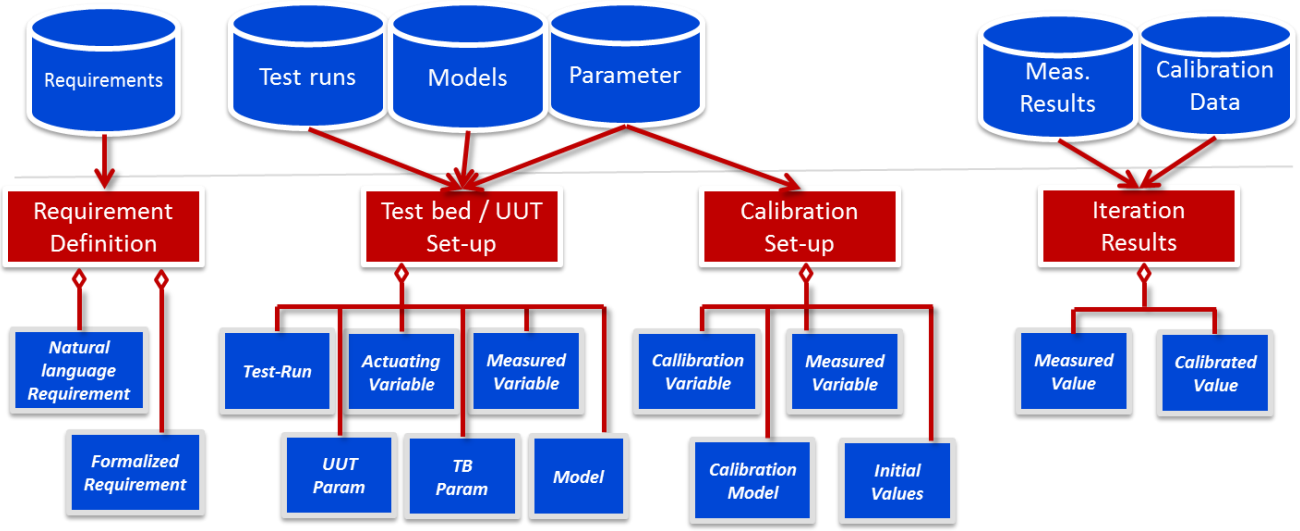

Figure 2-7: Associate Test and Calibration pattern parts to data categories and data artefacts

In, Figure 2-8 these test and calibration iteration activities are illustrated in the context of the applied tools such as in needed by WP3.4. Simulation model for Mathworks Simulink and AVL Cruise are configured for a certain testing and calibration set-up. These models are then generating iteration results, which are stored (in case of measurement values) in a measurement variable management system (i.e. AVL Santorin), or (in case of calibration values) are applied on a calibration tool such as AVL Cameo.

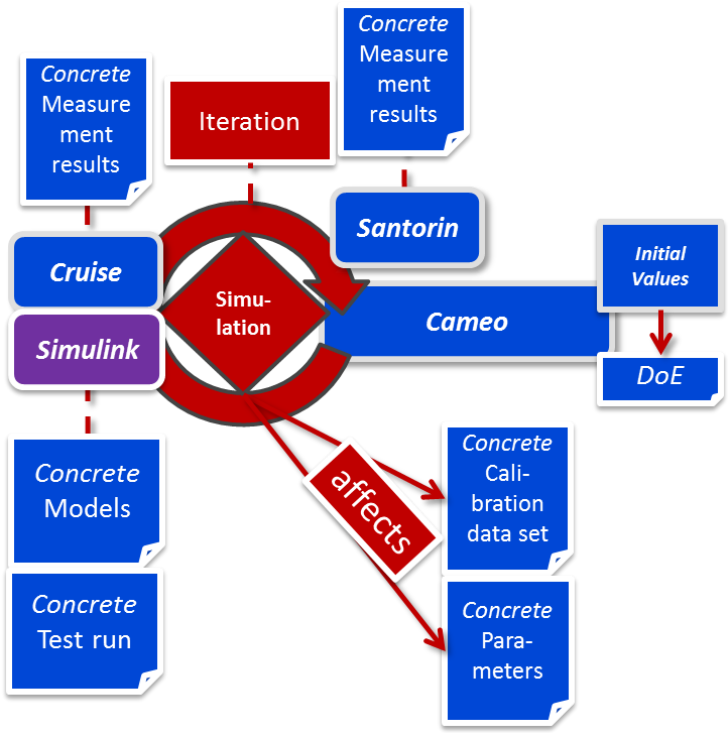

Figure 2-8: Test and calibration iteration in testing phase I (simulation)

In order to enable interoperability between these tools, an OSLC resource model has to be developed to integrate the different data categories and enable data navigation. These OSLC resource model is then applied to the AVL data backbone concepts.

Link to internal working documents:	

3 T6.13.2: MathWorks Simulink

3.1 Description

Name:	Mathworks Simulink
Contact:	selver.softic@v2v2.at
Dependencies	WP3.1, WP3.2, WP3.3 WP3.4, T6.3.5, T6.13.1, T6.13.4, B3.48
License	
Additional information	

MathWorks Simulink is a popular dynamic systems modeller with a broad scope of features, rich ecosystem and wide use. Simulink is found in many domains and is the source for complex model-based tool-chains in software development for embedded systems. MathWorks Simulink is the de-facto industry standard in simulation model development. Work on this brick will result in an integration of MathWorks Simulink and IOS in a manner that Simulink becomes straightforward accessible from other tools that comply with IOS.

3.1.1 About MathWorks Simulink

Powerfulness simulation tools such as MathWorks Simulink lead to great flexibility regarding the operation purpose of simulation models. As a consequence, simulation models can be used in very early as well as in very late development process stages. This, however, leads to the problem that some demanding characteristics and constraints of simulation models (such as simulation accuracy, real-time constraints, etc.) differs significantly in the various development stages and thus often hinder model-reuse and model development collaboration. Besides this, a lack of model development collaboration activities is still often found between different projects. Even if participants of the projects are aware of each other, there is often no straightforward access to the applied simulation model in order to analyse them regarding their potential of reuse. Analysis of coverage of use case specific needs summarized in following subsections define minimal required functionality that Simulink IOS adapter shall cover as follows: Adapter should support linking between Simulink models, Simulink blocks and stat-flows residing in systems like Integrity and the like with the Architecture models described in common languages like UML residing in tools like Enterprise Architect (EA), Papyrus etc, and allow the reflection of changes and pinpoint the occurring inconsistence between these models with less effort as until now. Further creation, update and deletion of requirements as well as linking to the Simulink entities shall be provided in order to enable tracking and notification mechanism on changes and keep the context of Simulink components up to date. Very essential aspect as well is managing the description interchange under provision of test cases and test scripts for different cases of functional testing and from different kind of data sources starting by simple excel sheets as parameter bases up to professional tools for these purposes. Security on protocol level is also an important issue that should be considered at least on the level of common web protocol. Amount and type of links and meta data that should be supported varies from use case to use case, which demands for technological infrastructure like OSLC (Open Services for Life Cycle) which is resistant on schema changes. The major objective is therefore to significantly improve collaboration and re-use of simulation models. The simulation models should be more straightforward accessible and findable in terms of their purpose to significantly improve project and model development collaboration.

3.2 Use Case coverage and application

3.2.1 Coverage in UC3.1

3.2.1.1 General Remarks

This section describes minimal set of requirements as they arise from the definition of use case 3.1 (UC 3.1)

The detailed description of UC 3.1 can be found in D301.010 and D301.011. The here discussed content does not surmount all requirements of UC 3.1 but only those which have effect on integration activities for Simulink.

3.2.1.2 Brief UC 3.1 Description

This use case addresses the development process at Volvo Trucks used when developing a new electronic architecture including vehicle functions. Volvo also investigates the use of behavioural modelling, both at early stages to validate requirements, and at later stages for software components verification and generation of code. The use case is thus a mix of current process at Volvo together with methods and tools that are interesting to investigate and may represent future possibilities. The purpose of the use case is to describe a comprehensive integrated development process although not all parts of it necessarily will be covered in depth within CRYSTAL. SystemWeaver is currently used at Volvo for requirements handling, functional design and early architectural and software design, like topology and decomposition of functional components into software components. The design made in SystemWeaver is then refined with Simulink.

3.2.1.3 List of Requirements

Use Case 3.1 requires the following functionalities in its use of Simulink:

1. It shall be possible to link a Simulink block or state-flow model to an architectural component in another tool.
2. It shall be possible to view requirements attached to the architectural component by selecting the corresponding block or state-flow model in Simulink.
3. It shall be possible to update requirements in Simulink (given that requirements are associated to Simulink entities).
4. It shall be possible to do deferred updates of requirements such that changes are notified but not automatically carried through.
5. It shall be possible to associate requirements to Simulink entities (given that the entity is linked to an architectural component in another tool and that there exist a set of requirements in a requirements tool).
6. It shall be possible to add new requirements in Simulink and store the created requirements in the requirements tool.
7. Any changes in architectural components and requirements in the design and requirement tools shall propagate to and be notified in the Simulink model (given that there are component and requirement links established).

3.2.1 Coverage in UC3.2

3.2.1.1 General Remarks

This section describes the requirements regarding the Simulink Brick B3.48 as they arise in use-case 3.2 (UC 3.2). The detailed description of UC 3.2 can be found in D302.010 and the latest progress on design in D302.011. The here discussed content does not surmount all requirements of UC 3.2 but only those which have effect on integration activities for Simulink. The set of requirements for the Simulink brick B3.48 and the set of requirements for the IOS will intersect but are not identical. Some requirements for B3.48 stem from issues of technical administrability, other from user experience and some of course from the logical organization of the use-case.

3.2.1.2 Brief UC 3.2 Description

UC 3.2 is concerned with the design and implementation of an on-board driving control system for series vehicles. This implementation is realized as a mixture of C-Code and Simulink models. The resulting system has to be designed under safety considerations but no safety norm applies to it as its application is restrained to certain artificial areas of use. The use-case owner has decided to follow the ISO 26262 as far as reasonable. Interoperability based on CRYSTAL's IOS is supposed to improve the consistency of

involved artefacts. The use-case also attempts to improve control over product variability and attempts to improve the efficiency of the V-cycle execution.

3.2.1.3 Context of Use-Case Requirements

The first concern of UC 3.2 is to improve execution of the work required by safety engineering. The ISO 26262-like design process requires a safety analysis, the estimation of risks and hazards, definition of resulting safety goals, derivation of safety requirements and finally a technical concept how to achieve those safety requirements (Technical Safety Concept / TeSaCo).

To current date, the TeSaCo is formulated in Enterprise Architect (B3.28) and is comprised of different UML diagrams for different purposes. Predominantly, the UML language is used to model chains of effect. The exact looks of the chain changes with implementation decisions if they affect the overall architecture of the solution. It must be warranted that such changes do not go unnoticed.

Since at the moment the architectural work is being performed using Simulink, the required safety concepts are not independent of a given implementation in Simulink. Changes to controller architecture modify the assumptions behind the TeSaCo. Such changes can be minor (renaming of objects) or major (logical units are replaced, routing of signals between blocks is modified, lines change logical meaning, etc.). As a result the use-case owner desires to get notified about changes to the controller architecture without polling the respective files. The means to achieve this is to set subscriptions to certain files and objects therein. Integrity can already support this as part of its strict workflows for files and for object representations mined from Simulink files but some additional intelligence is required in order to avoid unnecessary process steps like an ever repeating rechecking of TeSaCos. Moreover, if a TeSaCo was triggered for review then the review should be executed in the most informed way. The objects between the two models (UML and SL) should have a relationship and if a Simulink block was unilaterally renamed then the change of name should be somehow communicated to the reviewer or he will have to spend a lot of time to understand the changes.

It is probably possible to implement all this in Integrity alone but this will yield no improvements of interoperability. Given this specific issue it would appear beneficial if Simulink and EA could be prepared in such a way that they can exhibit new behaviour without having to change the logic of an Integrity configuration. It is important to note that the logical connections between EA and Simulink do not imply technical interoperability. Due to the choice to work with an advanced versioning and workflow mastering tool like Integrity, the technical interoperability between EA and Simulink is relatively thin. It would be only necessary if EA and Simulink had to interact on a local system, e.g. for what-if scenarios. Otherwise the logical relationships between Simulink and EA are satisfied if appropriate meta-data was made available via Integrity's database and Integrity was considered as a two-way high-latency communication channel between the two tools.

It must be differentiated between the old and the new process which is designed during CRYSTAL. In contrast to the initial development process, the new process amends that Simulink architectures and models shall be pre-designed with Enterprise Architect in order to close the gap between requirements and implementation. Such a gap reduction will help to better document and analyse the solution without having to deal with tentative implementations. It will also help to reduce the interdependence between Simulink and EA. Nevertheless, a substantial amount of code is present for which a quick re-engineering of UML is impossible. Because of this, the Simulink code must serve as reference to TeSaCos. This appears to be possible because the architecture of the solution has stabilized over time. Despite this fact, it must be assured that as soon as UML descriptions become available for a TeSaCo the safety engineer will be requested to allow for a shift of immediate reference to these intermediate models. It is one of possible measures to reduce linking efforts. Another measure is to automatically re-engineer appropriate UML descriptions using the new induction features of EA 10 and to establish links between source and UML.

How can the Simulink adaptor help in this? Consider a simple example: Not all changes to a Simulink file mean a logical change of architecture (as it would be relevant to a safety engineer). Modifications of files must be faithfully captured by a version control system, but their safety impact must be assessed on a more logical level. For example, a hash could be computed over important top level block properties (inports, outports, etc.) or the engineer could add indications that the safety concept must be revised.

In any review a mismatch between concept and model can be detected. In this case the Simulink model was ill-designed or the EA models were overhauled. In order to initiate a correction of either of the models either a special property in the model should be modified or some modification has to be achieved through Integrity. From experience, an immediate correction and resolution of discrepancies is not possible.

Therefore the ill relationship must be marked as such but should not appear as a new issue in the tracking system. If it was an issue in the tracking system then a semi-automatic impact analysis and overview over the current state of project description (D302.011 speaks of project cloud) will be much more difficult. Chances of repeated findings are high. What would be easy to evaluate is some kind of “stress connectors” which pinpoint inconsistencies, show into a direction of solution and contain educated guesses how to resolve them. Such connections could be automatically induced on Integrity’s side if Simulink sources were analysed upon receipt and additional workflow steps were introduced for the appropriate roles / institutions.

This requires some intimate knowledge about Simulink models that Integrity should not have for proper interoperability. Therefore, IOS adaptors for Simulink should be able to derive additional properties from models in a project specific way and these properties might depend on the current status of the workflow in Integrity. This means that some kind of dispatch/collect event handlers should be available for Simulink sources. A broader range of events can be imagined and they could be project specific or even specific to an object class. Such handlers will require some controlled project space or a managed local file repository.

The option to implement an agnostic Simulink-IOS adaptation seems technically easier but may appear inconvenient if the work activities (e.g. review) are not part of an Integrity guided procedure and the user has to separately login into Integrity (poses an interruptive additional step). Such an agnostic IOS adaptor will reduce the interactions between Integrity and the local Simulink copy to a simple check-in, check-out operations – which simply run over the IOS. These functions have to be provided anyway but it will mean poor interoperability.

Strong interoperability means a rich set of options to use a tool and parts of its logic for different purposes. An interoperability solution for UC 3.2 should allow triggering activities on behalf of remote applications. One such feature is mining of properties, simulation, modification of the models, etc. Transport of specific attributes into Integrity’s meta-data should be implemented by such an adaptor which can also contain work-flow controlling flags. Since not all applications are based on databases, providing functionality to the interoperability network also entails provision of temporal storage. This is similar to a “thread pool” but in this case the term “donor application” is chosen because installed applications will have to donate some of their functionality to the IOS-enabled development tool-chain. This might require cluster-like scheduling and coordination techniques.

The use-case owner has also required transferring properties like “time stamp of last modification”, “description” or “changed by”. The same IOS mechanisms can be re-used for different purposes, but it will require some introspection capability. Obviously, agnostic adaptors do not fulfil the expectation. A gnostic Simulink-IOS adaptation is technically more challenging. Every project is different and different rules will apply toward groups of logical objects and development artefacts. Hence, it will require some kind of project-specific configurability of the adaptors in order to control their behaviour and these configurations should be distributed in a consistent way across the project without much user interference. D302.011 has proposed a package service for IOS adapted tools in order to provide seaming-less experience.

The hereto discussed ideas are more related to IOS. Acceptance of IOS-integrated Simulink requires also consideration of usability which is not directly linked to IOS functionality - but can decrease IOS acceptance. A Simulink user should not have to leave his environment for many typical development events. This is standard approach with many frameworks using Matlab. It is assumed that prolonged interruptions of a developer’s activity are detrimental to his satisfaction, concentration and his efficiency. The Matlab environment allows seamless usability experience by providing integrations with the Simulink GUI and by allowing specific user-defined toolboxes.

An important request from the use-case owner was to allow intelligent types of linking as linking have been found to consume a lot of time. Without some computerized support linking can be also very error-prone. There are several logical types of links which are of interest. There links for indicating loose association (information, tagging, etc.), there are links for showing historical relationships between objects (C came from B which came from A), there are instructive links (refine, derive, transform, etc.) and control links (mutability, authorization, constraint information, etc.). Every of these links will be probably observed during UC 3.2 integrations. However, their technical implementation style may vary not only in the sense of change resulting from advancing IOS but also as a general principle of a project. For example, a link of the type “item implements requirement” could be at first an implicit link (e.g. implied from naming) and in the end be an explicit link (e.g. verified by user and stored in a file). Another example: An initial link could be established between two abstract sets (requirements group and superblock in Simulink). This link implies that all items in the sets are interconnected. The relationships could be fine-tuned with specific link suppressors for individual pairs. A single action will delete all implied links.

The two examples are given in order to indicate how linking efforts in a project could be minimized. Most of the links should be inferred from rules and powerful implication which will require some intimate knowledge about the modeller (here Simulink) and the PLM processes (represented in Integrity). The philosophy of the use-case is that 80-90% of such automatically induced links can be correct and the remaining links can be corrected by humans after errors were detected (add missing links, prohibit induced links, override specific attributes, etc.). This can yield a 10x efficiency increase from the perspective of the tool-chain user. Smart select, search and click connectivity for the complete set of objects in the development context is also requested. The technical choice of links must support this strategy and the choice must be supported from IOS and Simulink adapters.

What kind of technical links will be required to help realize the linking strategy of the use-case? The first question is what kind of links can be selected from. The following list explains how links can be technically categorized. Behind these categories different aspects of technical implementation reside:

direct link:	centralized link resolution (assured object identity)
indirect link:	distributed (e.g. via cascades) link resolution (volatile object identity)
absolute link:	no ambiguity of link (singular search result warranted)
relative link:	multiple ambiguity of link (result requires additional info -> drop some findings)
explicit link:	life-cycle is known (singular creation / destruction events), explicit link attributes
implicit link:	like-cycle is hard to predict, link attributes are inferred
managed link:	logical constraints in graph are enforced
unmanaged link:	logical constraints in graph are not enforced
permanent link:	link with low mutability rate (adequate for cold storage, difficult to access)
volatile link:	link with high mutability rate (requires hot storage, DB concepts, active systems)
solid link:	transparent link attributes (attributes are stored along with link)
proxy link:	opaque link attributes (attributes must be farfetched for link)

The categories can be mixed. Links can be direct or indirect, relative or absolute, explicit or implicit and they can be managed or unmanaged. The communication of links using OSLC is based on URIs which are an example of an explicit direct link which can have managed or unmanaged nature. Links can be also typed and underlie an extended taxonomy of links which will have a logical effect on constellations between other objects. Unmanaged links fail to respect logical constraints in the so formed graph. It's the linker's sole responsibility to link and unlink properly.

The CRYSTAL proposal has obligated the project party to base IOS on OSLC and this implies to re-use OSLC linking principles. URIs are the referencing mechanism of OSLC for identifying objects on a global scope. They are explicit, direct and global. Only within an RDF statement relative URIs are possible but they do not mean that relative linking is provided for objects existing outside those statements. Whether the respective URI was created under controlled circumstances or whether it has been established by a human is not visible to a system using the link. Every link using URIs in a respective RDF can be modified by the copy-holder at will. The application is responsible for not vandalizing the structure, to make the required modifications and to return the structures as RDF to the service providers. This assumes a high level of benevolent cooperation.

Unfortunately, for the needs in this use-case (and probably other use-cases) the Simulink integration will have to use indirect links in order to refer to items within same project increment or variant. Because Simulink will store all data about the model in an mdl-file (it is an XML file) it would be a great simplification if the file did not have to change content when a new project increment arrives or when it is reused in a new variant. Of course re-use could be achieved by rewriting the links upon check-out by Integrity's interoperability adaptor but it will disturb all tools in the tool-chain which check the files for change at the binary level. If indirect links were provided then there should be means to acquire a direct link given some context. This context could be maintained on the client side (Simulink interoperability adaptor) and it could provide a local proxy service with temporally limited explicit links for immediate use.

The use-case also requires an improved workflow in situations where design, implementation and testing are run concurrently and cannot finish before new implementation improvements, bug reports or change

requests are filed. It seems not to be practical to demand perfect consistency in the project but it seems necessary to demand a self-correcting process which achieves consistency if the number of open changes decreases close to zero. A good model for this is the avalanche model for the V development cycle. Every update to a piece of project data will cause further changes in adjacent V-steps until all adaptations yield zero change. In this case an implementation in Simulink has only to become consistent with requirements derived from the same avalanche. Normally, there exists no ready-to-use developer event to trigger such avalanches. The performed check-ins to Integrity's repository (and the resulting revision numbers) do not represent a coherent flow of corrections. They are very often containing partial results or additional defects which are simply stored for data securitization purposes. PTC has identified this problem and as a consequence Integrity provides an additional concept called Changes Sets (CS) which are meant to represent an avalanche of changes. The advantage of such change sets is that engineers working in different parts of the V-cycle will only get to see changes to documents if they have become really relevant for their work. This way they can pursue their tasks until a new project increment has become consistent enough for them to deal with it. For Simulink avalanches are irrelevant only to the point where a reference is taken to such CS. The developer should know which CS is responsible for the file he is editing and he should be able to cancel the CS (abort the avalanche) if he has acquired knowledge that the following CS will inevitably invalidate results of current CS.

Change Sets are another dimension to the project which is isolating changes and work-groups from each other. A changed Simulink implementation will cause a set of new test runs but these test runs should not become visible for later avalanches. Why this? Every new CS is something like a new project version and in general backporting and foreporting of code changes is difficult – especially if they face evolution of requirements. For example, if a bug was found on an older implementation and if it has to be fixed then it must be possible to do so without having to release completely new software. You can do so either by branching (status quo) or by working with avalanche barriers (new idea). The advantage of avalanche barriers is that it does not introduce variants (as branches) which do not exist in logical sense. The transport mechanism between avalanches must be the issue tracker. It should be easy to signal from Simulink that the modification has to hop avalanches and to become an entry in Integrity's issue management.

UC3.2 is demanding improved efficiency for the handling of variants. Most of the variability control will be realized with Integrity (B6.05) which is taking care of making the Simulink files available to the developer and to keep them versioned. However, Integrity is not performing logical checks on the check-ins so that locking of single files is only a weak instrument for maintaining consistency. The IOS adaptor could be extended to support stronger consistency rules. For example it could be enforced that the change of an inport in Simulink will also require a lock on the requirements and documentation documents related to that inport and that a check-in is only accepted if all changes can be executed atomically as a transaction for all involved items. Rich linking architecture can help to identify the necessary artefacts for such a transaction. In order to make this possible IOS should be able to reserve objects and to provide at least some multi-stage pseudo-transactions.

3.2.1.4 List of Requirements

The following table collects an initial list of requirements:

ID	Properties	Requirement
R061302000	nice	Assume flat network (e.g. no NAT, no configurable routers)
R061302010	nice	Not all machines own a global DNS entry. Provide means to communicate objects.
R061302020	must	SL-Workstations enter and leave in authorized way. Workstations can be certified to conform to project standards.
R061302021	must	Exchange of information between SL-Workstations can only happen after authorization. (Is not the same as R061302300 because this is at the workstation level. R061302300 is at the level of individual transactions. An authorized workstation can have been manipulated and in this case R061302300 will catch this.)
R061302022	should	A distributed authorization is applied.
R061302023	should	Allow overlapping authorization schemes (distributed/centralized).

R061302030	should	IOS-enabled SL-Workstations announce themselves in a global pool.
R061302040	should	IOS-enabled SL-Workstations will be triggered by Integrity to prepare for a new task
R061302050	should	Before a SL-Workstations becomes reconfigured for a new task the user is informed and he can control the exact point of time when his workstation will be reconfigured (e-mail, e-mail response or via dialogues)
R061302060	should	The user of an SL-workstation does not worry about keeping the file versions apart as long as his workstation is under control of a work-flow management tool.
R061302070	should	The SL-workstation uses additional information about the project in order to improve consistency strength of checked in modifications
R061302080	should	The SL-workstation can list relationships of objects.
R061302090	should	Frequent activities between project and SL should not involve complicated handling of windows.
R061302100	should	There should be relationships between model files which express the concept “refine”.
R061302110	should	SL supports a temporary “reverse create subsystem” function which maintains all linked relationships.
R061302120	should	SL-Workstation can mine models into RDF at arbitrary level of depth.
R061302130	should	SL-Workstation can compile RDF-descriptions to models and generated objects should inherit all links or original RDF objects.
R061302140	should	Selected objects in SL appear active in workstation’s context.
R061302150	nice	Models in context can be one-clicked for FMI interactions.
R061302160	should	Models in context can become visible to other workstations upon user request.
R061302161	should	Published models from R061302160 can be easily compared with each other.
R061302170	must	Developer is warned when he edits or deletes parts of the model which are part of safety argument (like from the TeSaCo)
R061302180	should	Certain model blocks can require additional formal reviews which are enforced during check-in.
R061302190	should	SL-Workstation can analyse models for specific kind of changes and set special flags on the checked in model in order to activate call-backs for subscriptions. E.g. send e-mail or start work-flows.
R061302200	should	SL-Workstation can handle implicit links like if they were implicit links.
R061302201	should	Manually modified implicit links become explicit links which override implications (also deletions).
R061302210	should	SL supports “expected datatype” links.
R061302211	should	SL will warn about unexpected typing of ports, lines, etc.
R061302220	nice	SL links blocks historically (this would simplify merging of changed models)
R061302230	should	SL provides a measure of how severely the model has been modified. These values become model properties stored in Integrity (or similar tool).
R061302240	must	Every check-in requires a comment. Prefill the comment with reasonable descriptions.
R061302250	should	SL-Workstation provides rich text developer documentation according to language settings.
R061302260	should	Developer receives a list of documentation artefacts which should be changed according to the changes observed in the model.

R061302270	should	Developer receives a list of tests to be adapted according to changes observed in the model.
R061302280	should	Developer is warned by the SL-workstation if some items were modified which are marked for “enforced broader consistency”. He can reserve or check out these objects and trigger their check-in as a network-wide transaction.
R061302290	must	SL-Workstation will extract meta-data from models and provide them to the network.
R061302300	must	SL-Workstation will not accept suspicious models or code which is not credibly signed.
R061302301	must	SL-Workstation providing not credibly signed models will be banned immediately. In this case SL-Workstation must immediately indicate visually that it has been banned (code BLACK)
R061302302	must	Banned SL-Workstations can be allowed by force. In this case SL-Workstation should show some visual signal that workstation is unsafe (code RED).
R061302310	should	SL-Workstation must be capable to operate even after a network outage (laptop operation, network failures, firewall activities, etc.). This means that some project descriptions should be held available locally (cache).
R061302320	should	Selecting object in context should highlight objects in SL.
R061302330	should	Developer can record an Instrument (cf. D302.011) (something similar to makro) for activities performed in SL.
R061302340	should	SL-Workstation can execute models upon requests via IOS.
R061302341	should	SL-Workstation can expose via IOS the records obtained during simulations.
R061302342	should	SL-Workstation can configure the model upon requests via IOS.
R061302343	should	SL-Workstation can record the results of simulations (e.g. expose workspace contents) and make them available to developer context. Errors are exposed as error objects. Results are chained historically.
R061302344	should	Step-by-step execution of simulation for debugging or simulation purposes. (It’s known that there are limits with what can be done with SL) Triggers come via IOS.
R061302350	nice	SL options are activated / deactivated given the work-flow step. Prevent specific changes.
R061302360	should	Read / write blocks for parameters and signals for RDF
R061302370	should	SL-Workstation can report installed toolboxes and their versions
R061302380	should	SL-Workstation detect another SL-workstation with MXAM or V&V and execute a static model test and return the results to the developer.
R061302390	must	SL-Workstation can generate various kinds of model views according to OSLC specification
R061302400	should	SL-Workstation should provide a regular web-page on port 80 which describes the status of the IOS-node to all staff using regular machines.
R061302410	nice	SL-Workstation support CBR-approaches. The system can find the most similar variant given the developer context.
R061302420	nice	SL-Workstation can support additional instructions or information given the developers context. This could be a warning or additional instructions as balloons. Rules to trigger the messages and the messages themselves are stored as objects in a central project representation (e.g. in Integrity).
R061302430	should	SL-Workstation can display remote OSLC-based user interfaces, e.g. when accessing EA models.

R061302440	should	SL-Workstation should offer choices about which Instrument is to be used if ambiguity is present. E.g. code generation for the same platform can be achieved either by Embedded Coder or by Real-Time Workshop.
R061302450	should	Optimize speed of Instruments. If it is possible try to make them available locally before executing them.
R061302460	must	SL-Workstation can shelve developer context and switch to another project without check-in.
R061302461	should	Local shelves are encrypted (something like TrueCrypt).
R061302470	must	Inspect complete history of changes to a given block or line.
R061302480	nice	Generate Matlab scripts from RDF descriptions.
R061302490	should	SL-Workstation can visualize current work-flow as it would be from Integrity (or similar tool).
R061302500	nice	Upkeep synchronized representative image masks for blocks with their content.
R061302510	nice	Provide program interactions for advanced input to blocks (e.g. a sophisticated filter designer could be used to enter a Laplace expression in Matlab fields).

3.2.2 Coverage in UC3.4

3.2.2.1 General remarks

This section describes minimal set of requirements as they arise from the definition of use case 3.4 (UC 3.4)

The detailed description of UC 3.4 can be found in D304.010 and D304.011.The here discussed content does not surmount all requirements of UC 3.4 but only those which have effect on integration activities for Simulink.

3.2.2.2 Brief UC 3.4 Description

In the Crystal project, AVL participates with two use cases (UCs) reflected by the WP3.3 and WP3.4. This chapter is all about WP3.4. It consists of several sub-use cases, which are, however, related to each other and have partially overlapping content.UC3.4a is led by AVL Graz and is about the interoperability challenges for vehicle testing scenarios at different vehicle development stages. AVL Regensburg is leading UC3.4b which is mostly about integrated tool environments for embedded control development and improvement of the development via an efficient variant handling within the V-cycle. AVL uses Simulink as part of their testing facilities in various vehicle development, vehicle function development (for instance ECU control software development) and testing phases.

3.2.2.3 Context of use case requirements

Development frontloading by early vehicle simulation and calibration iterations as well as re-using of these simulation and calibration results in later development phases are key challenges of this use case. All challenges have in common that corresponding methods and infrastructure for a sophisticated simulation model and data management has to be applied in order to establish traceability between all essential data artefacts.

Variants of a classical representation of a development V-model add the additional aspect of virtualization of the implementation in form of simulation models (e.g. of an engine). These simulation models make validation of certain design decisions possible at an early stage - also called development frontloading. Development frontloading enables early comparison and validation of different designs. Independent of that, the goal of or the story behind these models is the same: Building a vehicle.

A reasonable enhancement of the V-model in that manner illustrated above in form of a so-called W-model is illustrated in Figure 3-1.

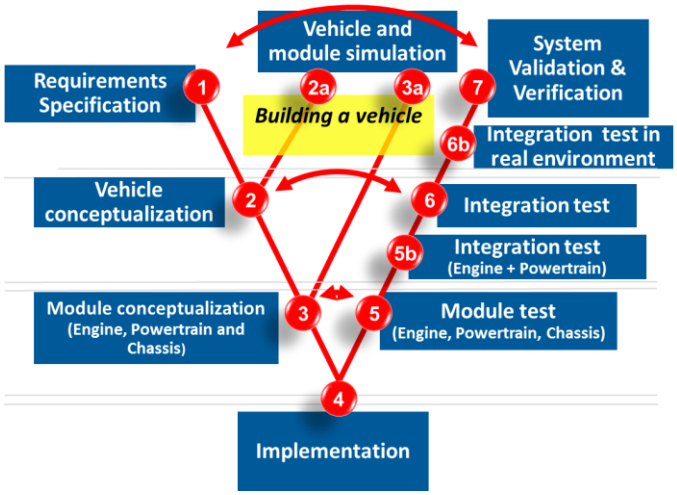

Figure 3-1: Early verification through frontloading W Model

As shown in this figure, requirements validation and verification becomes possible already at the *vehicle* or *module conceptualization* phase (2 and 3) by *vehicle and module simulation* (2a and 3a). In addition, rest vehicle simulation remains important on the right side of the W-model and is applied on so-called integration tests.

One example of using such (rest vehicle simulations – also with Simulink Models) is so-called test and calibration iterations applied on a certain unit under test (UUT). A general pattern of a test environment is about UUT calibration as shown in Figure 3-2. The test bed set-up configures the testing environment (with further sub-modules such as needed simulation models including their configurations, etc.), while the calibration set-up is specialized on tuning selected parameters of the UUT in order to fulfil the given set of requirements. The specification of these set-ups with all their sub-modules finally leads to the implementation and integration of the overall test set-up, which can be partly based on the simulation of vehicle and environment models (implemented by Simulink models).

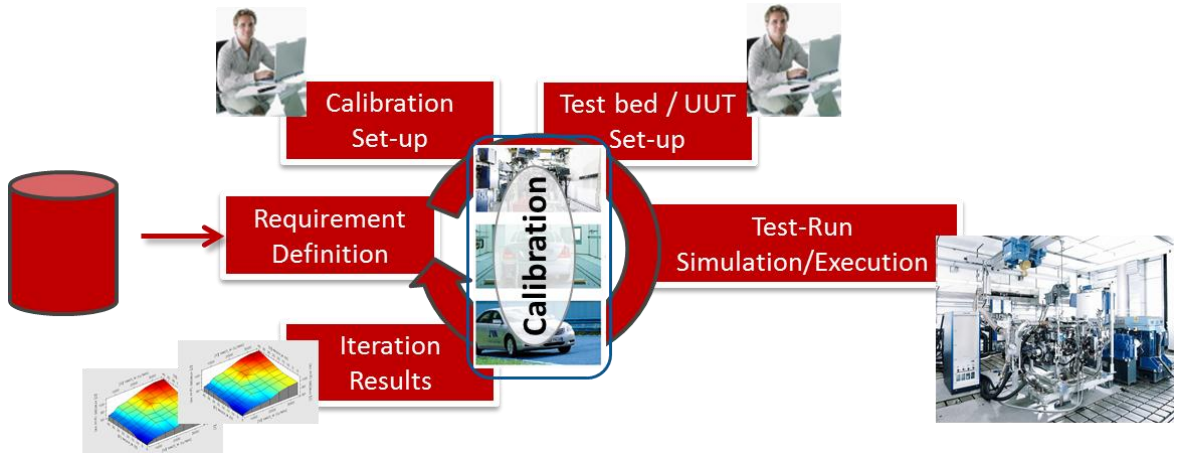

Figure 3-2 Test and calibration iteration pattern

Another example of using Simulink models is illustrated in Figure 3-3. Heterogeneous simulation (also called co-simulation) is the ability to couple two or more simulation models executed in different tools at run-time. The FMI standard currently evolves itself to be *the* standard for co-simulation (so far there has been none).

Figure 3-3 illustrates an example from the automotive domain: A rather overall vehicle model, which is embedded in an adequate vehicle environment model, and a more detailed engine model should be co-simulated. Two different simulation tools may be involved here, whereas each is specialized for its domain. For instance, the motor model is about a software model that consists about the control algorithms of the engine's ECU. Some of these models may are implemented in Simulink, while others are not (e.g. using AVL Cruise). Through co-simulation, however, combining Simulink models with other types of simulation models become possible.

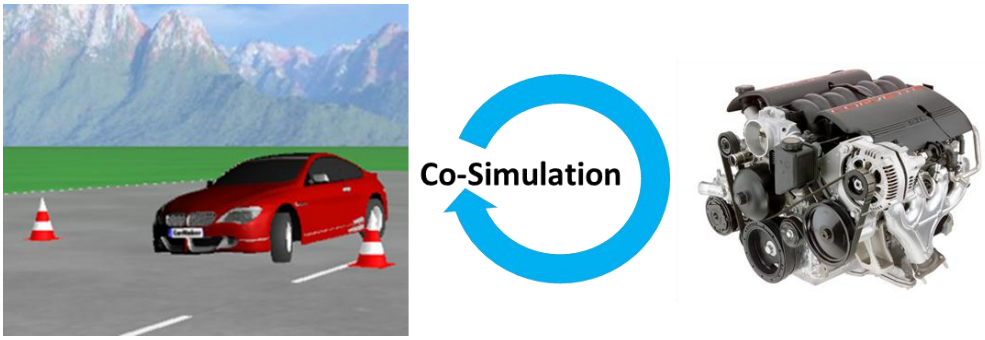

Figure 3-3: An automotive example about the co-simulation of an overall vehicle and vehicle environment model and a more detailed engine model

WP3.4 consists of two sub use cases. More than just being two separate instances of the common base UC, both variants should be combined as well in the following way: Since both use case variants are heavily based on simulation models, whereas the same, similar or related models are applied in both use cases (or are at least interrelated by certain simulation model parameters), the implementation of an appropriate Simulation Model Data Backbone (see brick No. B3.83, which is in tight relationship with Task 6.13.1) is essential to share certain models between the use cases. The simulation model data backbone should be attached to the appropriate tools in the several use cases as well under consideration of the principles of the interoperability standard (IOS).

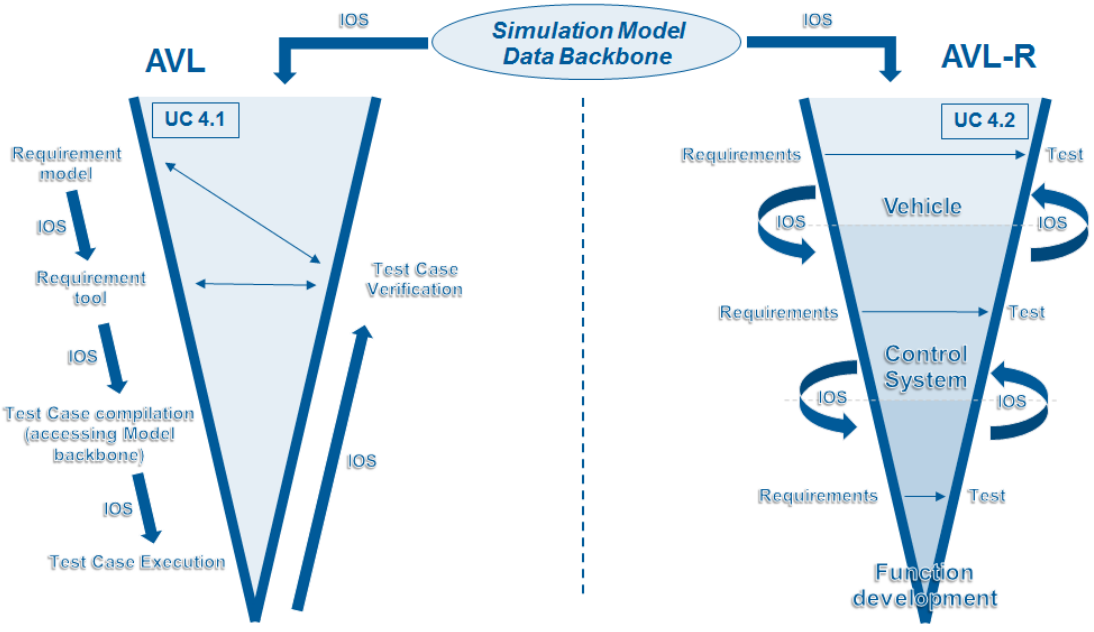

Figure 3-4: Model exchange between two different development process via IOS interfaces.

3.2.3 Coverage in UC3.5

3.2.3.1 General remarks

This section describes minimal set of requirements as they arise from the definition of use case 3.5 (UC 3.5)

The detailed description of UC 3.5 can be found in D305.010 and D305.011.

3.2.3.2 Brief UC 3.5 Description

The objective of the use case is to define the model for the concept of an automotive climate system, potentially safety critical, considering the functional safety constraints (ISO 26262) and the functional needs. The use case will be implemented according to the objective using the specific, real-world use case of a climate system (HVAC) with the indicated safety critical implications. Within the use case a Simulink model is going to be built for simulating the expected performance.

.

3.2.3.3 Context of use case requirements

According to Use Case 3.5 and more generally for all the use case in our applications, the role of Simulink tool is to model the physical/technical system, getting in input numerical data from usage scenarios (e.g. vehicle speed values during test or simulations of driving cycles) (text, excel format) and producing in output the calculated results related to the expected performances data (text, excel format).

3.2.3.4 List of requirements

Integration and interoperability are related to the tracing of changes in relationship to the functional safety modelling of the system, specifically with respect to SysML in the context of Enterprise Architect environment tool. A possible interesting evolution could also be the tracing of model sets for every variant of the system.

More in detail, it is possible to distinguish the following steps for integration and interoperability:

1. Link a Simulink block to an architectural component in SysML (Enterprise Architect) and vice versa.
2. Update requirements and parameter values associated to Simulink entities on the basis of the corresponding update of SysML architectural model requirements.
3. Associate requirements to Simulink entities from an architectural component in SysML (Enterprise Architect).
4. Propagate and notify any changes in architectural components and requirements from SysML (Enterprise Architect) to Simulink model and vice versa.

3.3 General Improvement

3.3.1 Improvements for WP3.1, WP3.2, WP3.4 and WP3.5

No special tool improvements needed except those described in Section 3.4.

3.4 Integration and Interoperability

3.4.1 Improvements for WP3.1

TI NAME: Tracing and linking infrastructure functional block to connect Simulink block or state-flow model to an architectural component					
TI_ID	CRYSTAL_TI_6.13.2_1	**Kind of TI**		**Contact email**	Daniel.b.karlsson@volvo.com
Description: Main objective of this component to provide infrastructure to create and maintain links between architectural components and Simulink model which are involved into use case workflow. Hereby it should be kept in mind that models delivered will be most probably in EAST-ADL. In order to support this functionality for the IOS coupling and at the same time stay conform to other TIs an OSLC Provider					

at each side which supports full CRUD (Create Request Update Delete) operability via REST Services should be implemented. In this way the links can be generated and updated as well delete if there is no need for them. Linking the models and architecture provides also desired traceability since OSLC delivers data also querying capabilities. Description of meta data domain is supported by OSLC AM (Architecture Management) vocabulary specification in RDFS (Resource Description Framework Schema) for the case of architecture documents while for Simulink models a common schema based upon RDFS can be consolidated with other TIs. OSLC supports both: own defined specifications as well self-defined vocabularies (data meta models). Additionally to the OSLC Providers as part of overall OSLC Adapter OSLC Consumer for interlinking and tracing the requirements connection from the CRYSTAL_TI_613_6.13.2_2 should be integrated as well. Data stores in behind can be commercial (IBM Jazz or Dassault) or open source implementation like Jena file based stores.

Link to internal working documents:	

TI NAME: Adapter for tracing, creation, maintenance and linking of requirements to architectural components and Simulink					
TI_ID	CRYSTAL_TI_6.13.2_2	**Kind of TI**		**Contact email**	Daniel.b.karlsson@volvo.com
Description: Main intention of this component could be of common interest for IOS in general, since it is specified only in conceptual level. The functional block represented by this TI should provide full operability over requirements repository offering tracing and maintenance of requirements and their links to architectural concepts and Simulink models. This component will be most probably realized with OSLC RM (Requirements Management) Specification as meta data and interlinking schema which is expandable and easy to combine Linked Data and RDF(S) technology. As solution a OSLC Adapter consisting out OSLC Provider for requirements which supports full CRUD (Create Request Update Delete) operability via REST Services. Data stores in behind can be commercial (IBM Jazz or Dassault) or open source implementation like Jena file based stores.					
Link to internal working documents:					

3.4.2 Improvements for WP3.2

TI NAME: Adapter for exchange in workflow between Simulink and Enterprise Architect with involvement of PTC Integrity					
TI_ID	CRYSTAL_TI_6.13.2_3	**Kind of TI**		**Contact email**	ruediger.diefenbach@daimler.com
Description: Best technological approach at the moment out of the sight of IOS for this integration would be in implementation of OSLC Adapters containing a Consumer and Provider for both sides. Main meta data model should be based on OSLC AM (Architecture Management) QM (Quality Management) and RM (Requirements Management) Specification extended for specific purposes of the use case with additional light weight meta model specifications. In this way traceability would be complete as well as exchange of needed specifications and belonging artifacts without changing the configuration structure of the PTC Integrity as demanded by use case owners. Mining of properties, simulation, modification of the models, as well intelligent links as main demands would be fully or partly in a high range covered because OSLC resides on flexible schema (meta data mode) engagement. Some special use case specific adaptation would necessary though. With OSLC as standard and CRUD (Create Request Update Delete) features it would comply with other Simulink TIs and IOS vision in general. Since OAuth is a part of OSLC reference implementation many security issues described in use case specific needs would be at least at protocol layer covered out of the box.					
Link to internal working documents:	• PTC Integrity (T6.8.9)				

3.4.3 Improvements for WP3.4

TI NAME: Simulation model exchange across development phases and working groups					
TI_ID	CRYSTAL_TI_6.13.2_4	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com

Description:

In Figure 3-1, a general interoperability pattern with the use of OSLC on top of various data backbones is presented. Let's assume that there is an in-house data backbone (called in AVL Data Backbone in case of WP3.4a), which stores simulation models beside various other related data categories. In addition, however, further data backbones have to be considered, e.g. provided by third party tools (such as PTC Integrity in case of UC3.4b). Depending on the data categories stored in these data backbones, a meta-model structure has to be defined on top of the data backbones that abstracts from the detailed content of each data backbone. This means in other words that these meta-model is defined independent of the location of storage on the one hand and is limited to only those data entities (across the data categories) that need to be linked to each other.

The OSLC concept perfectly matches to that demand. So-called OSLC domains capture a certain data category and OSLC resources reflect data entities that need to be linked against each other. It is important to mention that these OSLC resource models should be defined as minimalistic as possible and should thus be driven by the necessity of data relations. Every detail of a specific data entity should remain to the tool that creates and edits these details and/or the data backbone that is used to store these details.

It is important to enhance MathWorks Simulink with adequate OSLC adapter to work within such an environment.

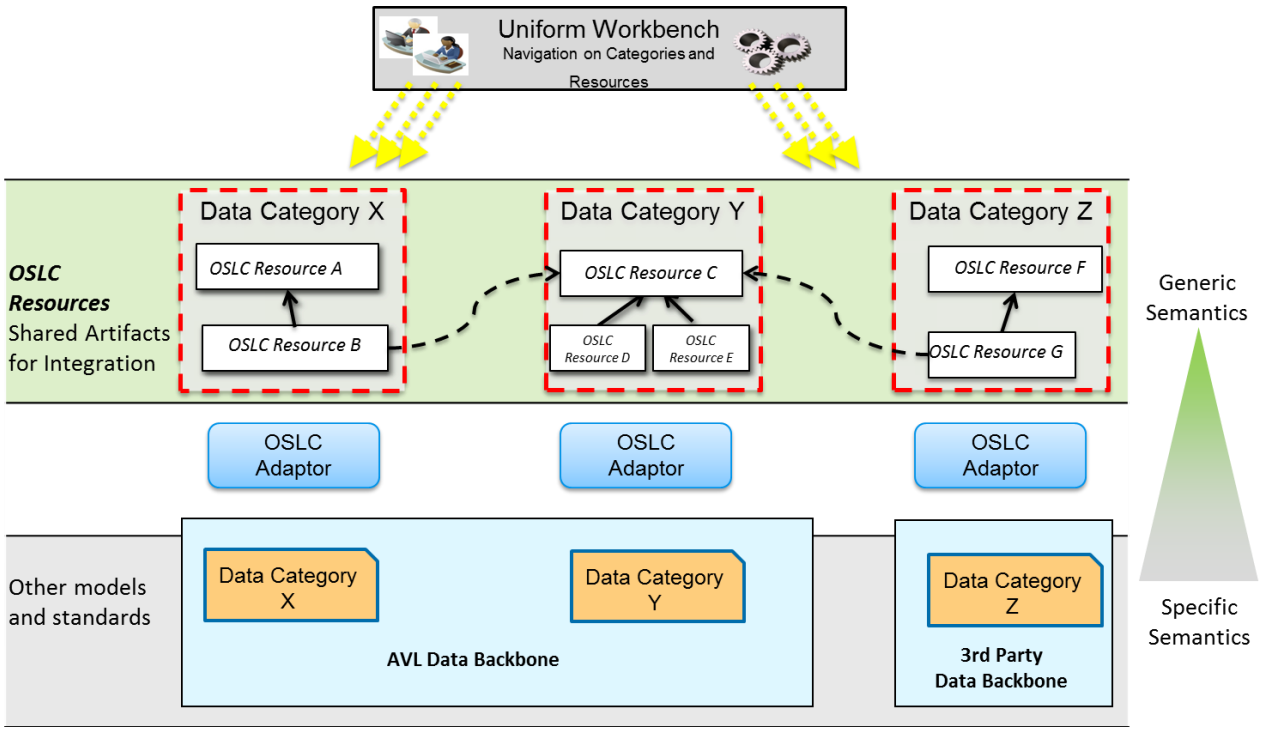

Figure 3-5: General OSLC pattern of the interconnecting data of different data backbones via OSLC

Link to internal working documents:	• AVL Simulation Model Data Backbone T6.13.1

TI NAME: Requirement Traceability to Simulink Models					
TI_ID	CRYSTAL_TI_6.13.2_5	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com
Description:					

Requirements and test cases have to be linked against further data artefacts that are verifying the requirements and are executing the appropriate test cases. In WP3.4, for instance, this aspect is especially needed for simulation models in form of Simulink models. Simulation models play an important role in most of the testing activities and phases and exchanging models across development phases is a crucial factor. It would be a significant shift for supporting model re-use if traceability between requirements and models elements (that are about to verify these requirements) is established consistently.

In Figure 3-6, a proper scheme for related interoperability challenges is sketched: An authoring tool (e.g. Simulink, AVL Cruise, etc.) is connected to an ALM Environment (e.g. HP Quality Center, PTC Integrity) via OSLC adapters. On top of these adapters an OSLC resource model addresses exclusively the top level elements such as requirements, test cases as well as models including their basic elements. Everything else remains to be interpreted by the authoring/ALM tool itself.

It is important to enhance MathWorks Simulink with adequate OSLC adapter to work within such an environment.

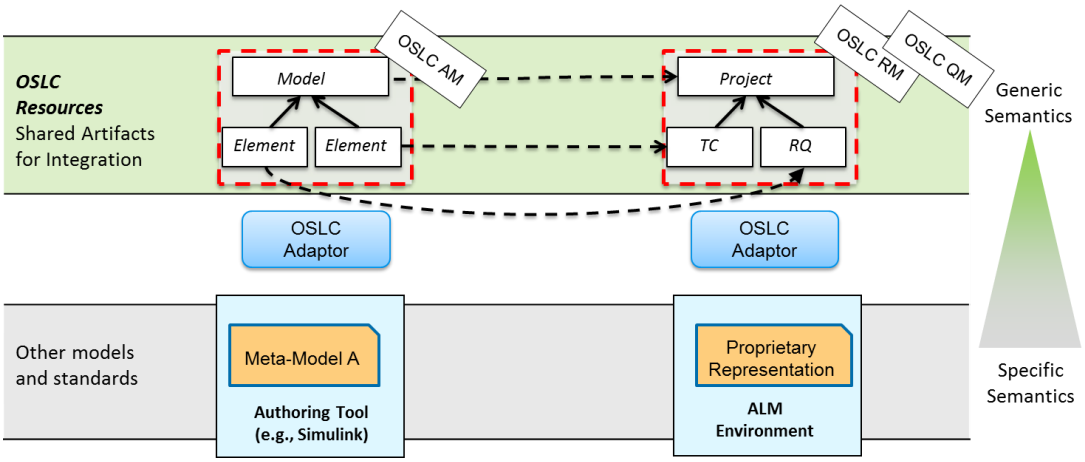

Figure 3-6: Towards an IOS OSLC scheme for interlinking authoring tools and requirements

Link to internal working documents:	• HP Quality Center (T6.8.4) • PTC Integrity (T6.8.9) • AVL Cruise (T6.3.5)

TI NAME: Co-Simulation					
TI_ID	CRYSTAL_TI_6.13.2_6	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com

Description:

Figure 3-7 illustrates an appliance of co-simulation for the task of calibration regarding testing a hybrid vehicle of sub use case UC3.4a. In the simulation phase, this hybrid vehicle is represented by an AVL Cruise simulation model. The model is calibrated by initial value and simulation is performed. The simulation results are then analysed by the calibration tool AVL Cameo as described by the calibration iteration engineering method in WP3.4.

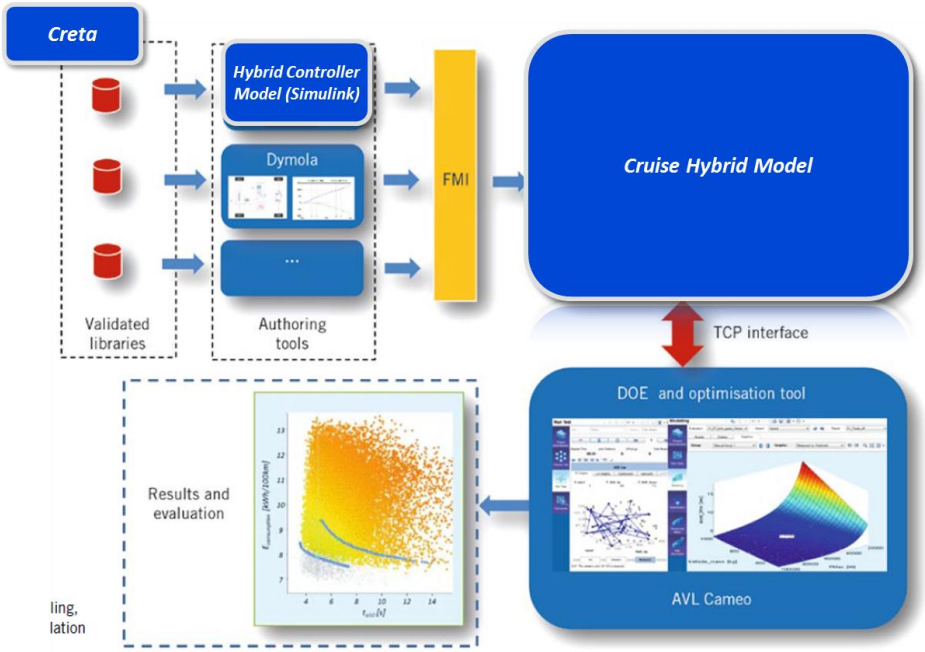

Figure 3-7 Possible interoperability scenario for co-simulation in the use case scenario

The next step in the vehicle V-model is to increase the accuracy of the simulation by a finer grained simulation of vehicle sub-components. The functionality of ECU controller is such a sub-component and Simulink models are wide-spread to model that functionality. In Figure 3-7, the Cruise hybrid model is thus extended by a hybrid controller Simulink model. The Cruise and the Simulink model should then interact at run-time via the FMI interfaces. In addition the hybrid controller model needs also to be involved in the overall calibration process. In fact, ECUs calibration is the most classical use case for calibration. Consequently, existing calibration values should be re-used from previous projects if possible. The AVL Creta tool is able to manage such calibration data and acts thus as an input source for hybrid controller model calibration as illustrated in Figure 3-7 as well.

It is important that MathWorks Simulink works within such an environment, especially with FMI-related topics.

Co-simulation has by nature nothing to do with the linked-data concept of OSLC. Nevertheless, in Figure 3-8, a useful combination of these interoperability types is possible: Independent of the run-time aspects and details such as which data ports of the models have to exchange data, the given fact that two models are related to each can be defined by OSLC links as well. Corresponding OSLC adapters abstract from the particular modelling tools and map the corresponding model to a OSLC resource model, which consists just of basic standard modelling elements representing a hierarchically structure and which model elements of a model A are related to what model elements of model B. Details of these model-interrelations such as the applied data types of concrete connected ports may remain on the responsibility of the FMI co-simulation standard.

Thus, OSLC adapters for MathWorks Simulink have to be adapted accordingly.

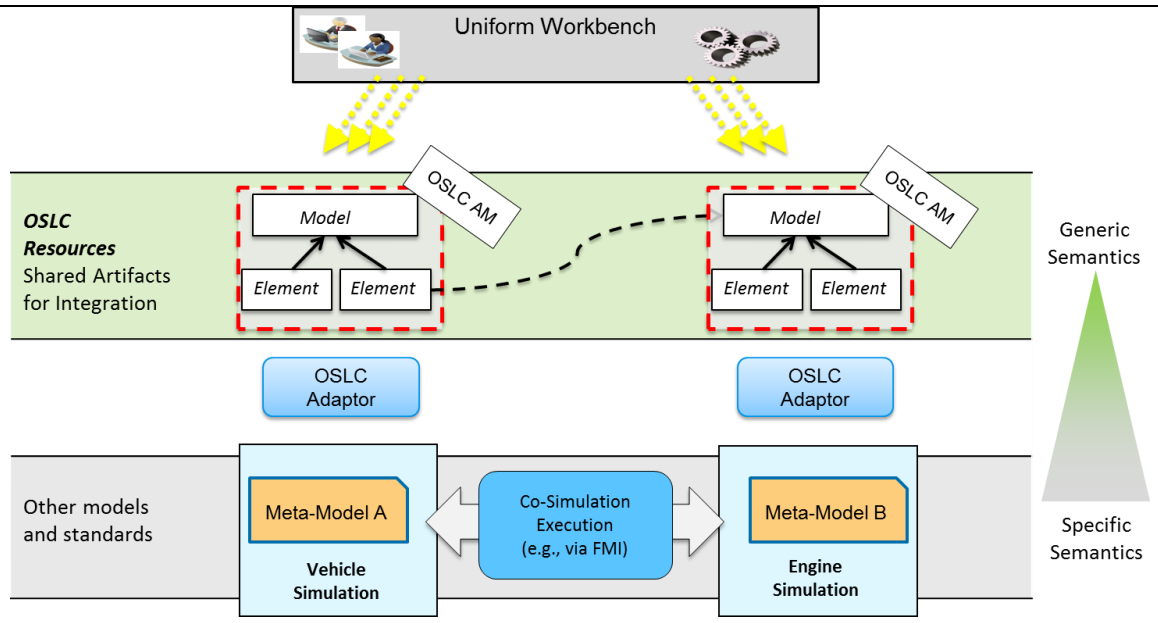

Figure 3-8: Combining the OSLC linked-data approach with co-simulation aspects

Link to internal working documents:	• AVL Cruise (T6.3.5) • AVL Creta/Cameo (T6.10.7)

3.4.4 Improvements for WP3.5

TI NAME: Tracing and exchange data backbone between Enterprise Architect and Simulink					
TI_ID	CRYSTAL_TI_6.13.2_7	**Kind of TI**		**Contact email**	TBD

Description:
The role of Simulink tool is to model the physical/technical system, getting as input numerical data from various usage scenarios in form of various sources and producing in output the calculated results related to the expected performances (both in text and excel format). Hereby links from Simulink block to an architectural component in SysML (Enterprise Architect) should be established and vice versa. An update of requirements and parameter values associated to Simulink entities on the basis of the corresponding update of SysML architectural model requirements should be provided as well. Requirements should be associated to Simulink entities from an architectural component in SysML (Enterprise Architect). All changes in architectural components and requirements from SysML (Enterprise Architect) to Simulink model and vice versa should be propagated and notified.
Out of the sight of IOS implementation of OSLC Adapters consisting from a Consumer and Provider would be appropriate in this case. Meta data model for input and output exchange could reside on OSLC AM (Architecture Management) and RM (Requirements Management) Specification extended with additional use case specific properties as external mini RDFS schema. Also the demand if traceability could be supported for tracking of

Link to internal working documents:	

4 T6.13.3: AVL TBSimu

4.1 Description

Name:	AVL TBSimu
Contact:	selver.softic@v2v2.at
Dependencies	T6.13.1, T6.13.2, T6.13.4
License	
Additional information	

Realtime test bench simulation based on AVL ArteLab©

The objective of Testbed Simulator (TBSimu) is to provide a consistent simulation environment for testing test bed automation and application development with realistic and repeatable measurement values. The number of adaptations if migrating from simulated test bench and real test bench shall be as small as possible.
For perfect results simulation is located on a separated hardware. For connection to automation system available standard interfaces are used. All values required for application development are delivered by simulation based on a consistent model.

Content of simulation
With TBSimu, an arbritary set of test bed components can be simulated (simulation of an entire virtual testbed)

- engine and load unit
- emission certification equipment
- measurement and conditioning devices

Simulation model:
Simulation models applied on TBSimu with the following features:

- up to 1 kHz simulation frequency for engine-/dyno simulation
- simulation of control values for an engine test cell.
- simulation of emission and fuel consumption, temperatures, etc.

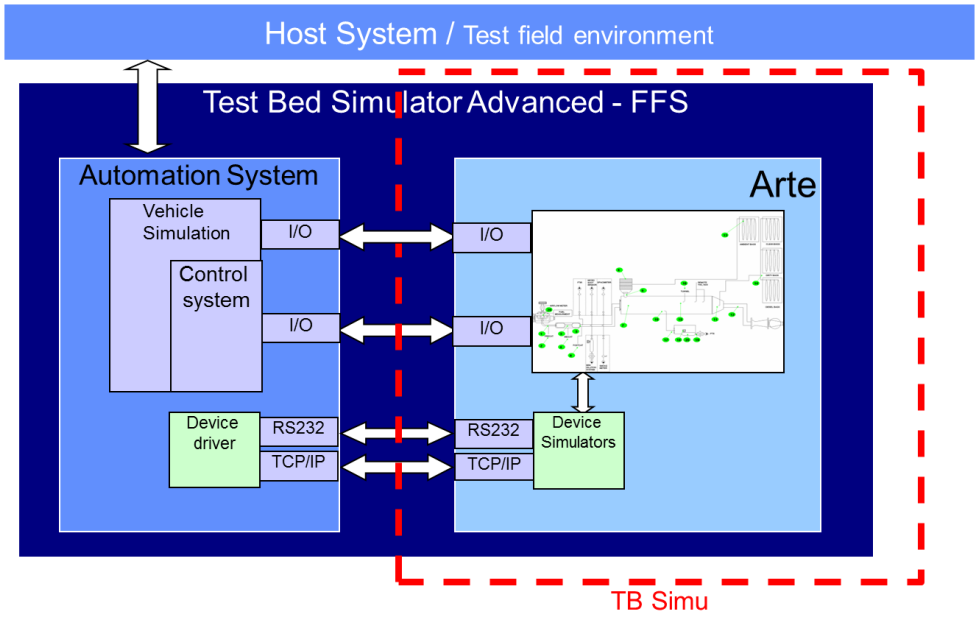

Figure 4-1: Hardware topology of a TB Simu system

Device simulator Software

Device simulators are connected like real devices via RS232 or TCP/IP. ASCII protocol is used.

Measurement results of devices depend on engine operating point and sample point. They are consistent over all devices.

4.1.1 Manual

For the overall goal of the SP3 automotive domain, developing a vehicle is the overall user story. Testing a vehicle is an essential part of the process of developing a vehicle and is an engineering method of its own. Creating an adequate set-up for a testing environment (test bed) is a complex assignment. A test bed simulator such as AVL's TBSimu supports the development proper testing environments by virtualizing test beds in order to speed-up test environment development by simulation frontloading methods. In that way, improvements and IOS integration of TBSimu is considered to support the SP3 domain accordingly.

4.2 Use Case coverage and application

4.2.1 Coverage in WP3.4

According to CRYSTAL's application document, the AVL TBSimu (Brick 3.73) is associated with WP3.4 (UC3.4a). At the current stage of UC definition, AVL TBSimu is not yet explicitly in the UC definition included, but is considered to play a role at a later use case definition phase.

The use case scenario of UC3.4a defines several requirements that adhere to the WLTP emission legislation specification. WLTP stands for World-wide harmonized Light-duty Test Procedure and is currently available as a draft specification. Besides certain testing requirements it contains also concrete standardized test-runs for emission testing. The TBSimu could support the creation of WLTP compliant test procedures by providing a consistent simulation environment for emission testing with realistic and repeatable measurement values

More detailed information will be provided in the next deliverable version based on the enhanced UC definition in WP3.4.

4.3 General Improvement

4.3.1 Improvements for WP3.4

Tool improvements may be needed to support the planned activities in WP3.4. However, details of these improvements cannot be considered in this deliverable, since AVL TBSimu is not yet explicitly included in the WP3.4 UC definition. More detailed information will be provided in the next deliverable version based on the enhanced UC definition in WP3.4.

4.4 Integration and Interoperability

4.4.1 Improvements for WP3.4

The main objective of this task is to improve the interoperability with other tools by applying IOS concepts in WP3.4. Most likely a tight interoperability with the Simulation Model Data Backbone (T6.13.1) will be needed. Furthermore, Simulink models (T6.13.2) have to be provided for test bed simulations as well. Since the role of AVL TBSimu is not yet defined in this work package, the integration concepts for IOS will be presented in the next deliverable based in the enhanced UC definition in WP3.4.

5 T6.13.4: AVL ARTE.Lab™

5.1 Description

Name:	AVL ARTE.Lab™
Contact:	selver.softic@avl.com
Dependencies	T6.13.1, T6.13.2, T6.13.3
License	
Additional information	

AVL ARTE.Lab™ Runtime Environment (RTE) is a run time environment to execute MATLAB® Simulink®-based real-time applications on AVL test beds. AVL ARTE.Lab™ Runtime Environment (RTE) is an extension to run MATLAB® Simulink® models generated with AVL ARTE.Lab™ Studio Software Development Kit (SDK) as real time application (RTA) on a PUMA / EMCON PC. The application can be started automatically with the PUMA Open automation system for test bed control (see Figure 5-1).

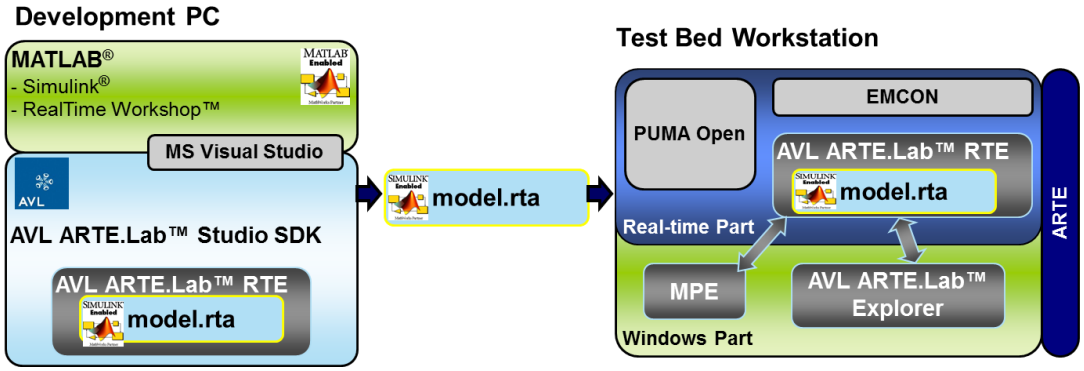

Figure 5-1: Overview of installed SW on development PC and Test Bed Workstation

In order to provide a comfortable and convenient environment for the customer-model specific (additional) parameters, the AVL ARTE.Lab™ RTE is provided with a generic Model Parameter Editor (MPE). This tool allows the smooth management of the customer model parameters (loading of pre-defined parameter sets on the test bed, online change of parameters). Due to the use of MPE, an additional MATLAB® license is no longer required.

Development of customer applications (RTAs) is realized on an external MATLAB® development PC with AVL ARTE.Lab™ Studio SDK.

The option of integrating MATLAB® Simulink® models into the test environment opens up a wide field of application for ARTE.Lab™ on the AVL test bed, such as e.g.:

- Simulation
  - Transmission models
  - Vehicle models
  - Test bed simulation
  - Driver models

  - Residual bus simulation
- Customer-specific control algorithms
- Integration of special filters
- Calculation of complex formulas
- Online measurement data analysis

AVL ARTE.Lab™-based real-time applications are fully integrated into the PUMA Open test bed system and automatically started and stopped like any other PUMA Open application. An authorized test bed administrator can integrate a customer-specific model by performing only a few steps. No special knowledge of models is required for further use on the test bed.

The AVL Real Time Environment (ARTE) processes the application in a time-synchronous manner and provides the PUMA Open system with all of the model's input and output quantities at ARTE's and/or the model's frequency. These quantities can be used like any other PUMA Open normname or system variable (PUMA Open Recorder, output on PUMA I/O, etc.).

In addition to online parameterization with the MPE, test bed system parameters can be read in automatically at runtime.

AVL ARTE.Lab™ Explorer is an AVL ARTE.Lab™ Runtime Environment (RTE) extension to visualize internal processes in models of MATLAB® Simulink®-based real-time applications. The Simulink® block output signals of all running models, the model I/O, the associated parameters and the model-specific data (frequency and calculation time) are displayed in a tree structure. From this structure, the user can select the signal/parameter of interest and show/change the related value.

All signals and parameters can additionally be displayed in an integrated multi-line graphic, which allows model developers to easily and conveniently test their real-time applications.

5.1.1 Manual

The execution of simulation models (e.g. environment models) is an essential part of vehicle testing. Depending on the development stage of the vehicle, not the entire vehicle is available by physical components. Then so-called rest vehicle component simulations have to be performed on the test bed.

ARTE.Lab™ is the link between such simulation models and vehicle testing. It ensures their execution on a proper vehicle test environment in order to speed-up the entire vehicle development process by the use of simulation frontloading methods. As real vehicle components exist on the testbed, the rest vehicle simulation models have to be executed under real-time conditions. In that way, improvements and IOS integration of ARTE.Lab™ is considered to support the SP3 domain accordingly.

5.2 Use Case coverage and application

5.2.1 Coverage in WP3.4

In UC3.4a of WP3.4 different vehicle test phases associated with appropriate vehicle development phases are a central point. In many cases vehicle testing means a combination of real physical components (driven by a test bed) and simulated components (rest vehicle simulation). Since ARTE.Lab™ plays an essential role in linking simulation models and such test bed environments, it is applied in various test phases described in WP3.4. On the one hand, ARTE.Lab™ should be able to handle different kinds of simulation models simultaneously (co-simulation) under run-time conditions, on the other hand model management (e.g. by an OSLC integration) should be supported as well.

5.3 General Improvement

5.3.1 Improvements for WP3.4

TI NAME: Support for different kind of simulation models for AVL ARTE.Lab					
TI_ID	CRYSTAL_TI_6.13.4_1	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com

Description: Currently, there are tools such as AVL ARTE.Lab™ Explorer and AVL Model Parameter Editor (MPE) which are limited to operate with simulation models created with MathWorks Simulink. Consequently, extensions have to be implemented to these AVL tools in order provide the possibility to use them independently from the authoring tool the simulation model was generated with.	
Link to internal working documents:	

5.4 Integration and Interoperability

5.4.1 Improvements for WP3.4

Currently ARTE.Lab™ focusses on enabling the execution of MathWorks Simulink models on real-time testing environments such as the AVL PUMA Open test bed automation system. In order to go beyond the support of a single simulation platform, further technical solutions are available at the moment. However, these approaches could or should to be improved and enhanced in two directions: On the one hand, it has to be evaluated if relations of simulation models on a high abstraction level lead to a better workflow. On the other hand, interoperability standards such as FMI should be used to enable co-simulation and thus simulation data exchange between simulation models from different authoring tools at run-time. These two aspects are reflected by the following two technical items:

TI NAME: Establishing simulation model relations with OSLC in a co-simulation environment					
TI_ID	CRYSTAL_TI_6.13.4_2	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com
Description: An example of combining OSLC with other standards is illustrated in Figure 5-2. Heterogeneous simulation (also called co-simulation) is the ability to couple two or more simulation models executed in different tools at run-time. The FMI standard currently evolves itself to be *the* de-facto standard for co-simulation (so far there has been none). This kind of interoperability has by nature nothing to do with the linked-data concept of OSLC. Nevertheless, a useful combination of these interoperability types is possible: Independent of the run-time aspects and details such as which data ports of the models have to exchange data, the given fact that two models are related to each can be defined by OSLC links as well. Corresponding OSLC adapters abstract from the particular modelling tools and map the corresponding model to a OSLC resource model, which consists just of basic standard modelling elements representing a hierarchically structure and which model elements of a model A are related to what model elements of model B. Details of these model-interrelations such as the applied data types of concrete connected ports may remain on the responsibility of the FMI co-simulation standard.					

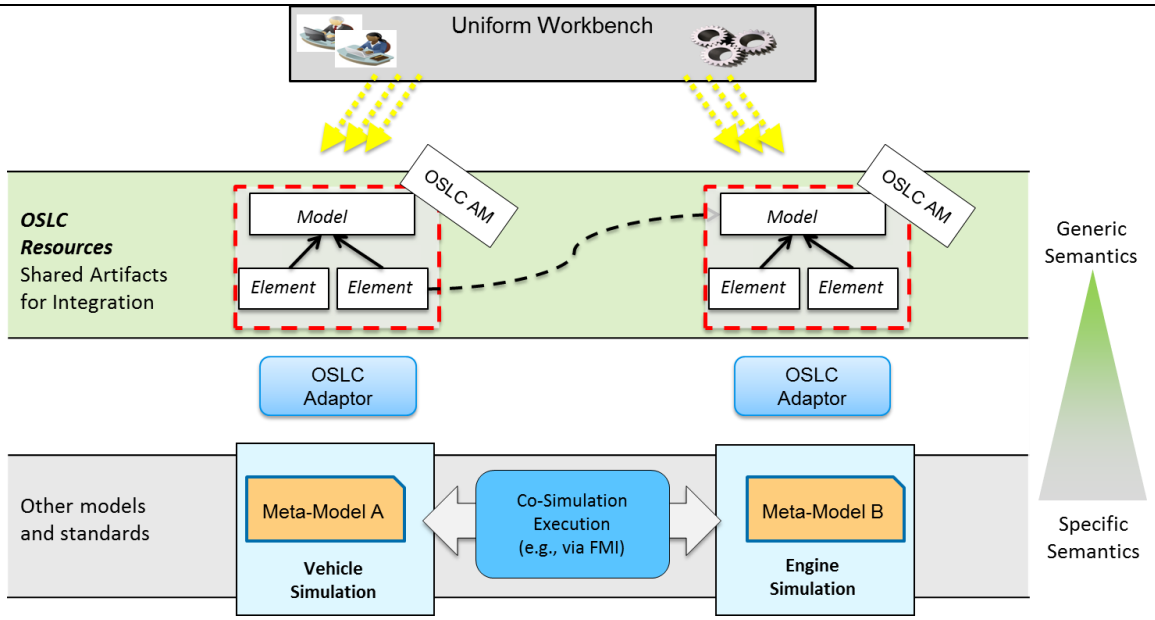

Figure 5-2: Combining the OSLC linked-data approach with co-simulation aspects

In case of AVL ARTE.Lab™ co-simulation between MathWorks Simulink models and other types of simulation models such as AVL Cruise models should be enabled. Cross-referencing between the models on an OSLC concept should be evaluated as sketched in Figure 5-2

Link to internal working documents:	

TI NAME: Establishing co-simulation for AVL ARTE.Lab™ with FMI					
TI_ID	CRYSTAL_TI_6.13.4_3	**Kind of TI**		**Contact email**	gerald.stieglbauer@avl.com

Description:
Figure 5-3 illustrates an example from the automotive domain: A rather overall vehicle model, which is embedded in an adequate vehicle environment model, and a more detailed engine model should be co-simulated. Two different simulation tools may be involved here, whereas each is specialized for its domain. For instance, the motor model is about a software model that consists about the control algorithms of the engine's ECU.

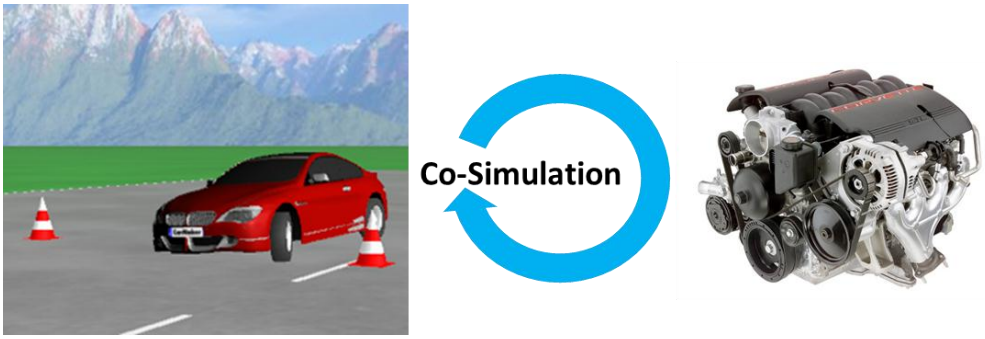

Figure 5-3: An automotive example about the co-simulation of an overall vehicle and vehicle environment model and a more detailed engine model

Figure 5-4 illustrates the use of FMI in the context of ARTE.Lab. Currently, tools such as ARTE.Lab Explorer and Model Parameter Editor are limited to models created with MathWorks Simulink. The integration of FMI in addition to corresponding extensions of these AVL tools will enable the possibility to use the tools

independently of a specific authoring tool the simulation model was created with.

The engine test bed is controlled on a real-time operating system on the one hand, and by non-real-time configuration tools running under Windows on the other hand. Within the real-time part, different kinds of simulation models are applied, such as AVL Cruise models and Simulink models. In the latter case, Simulink models are embedded into an execution environment called AVL ARTE.Lab™. During the CRYSTAL project, co-simulation of simulation models supporting FMI has to be ensured.

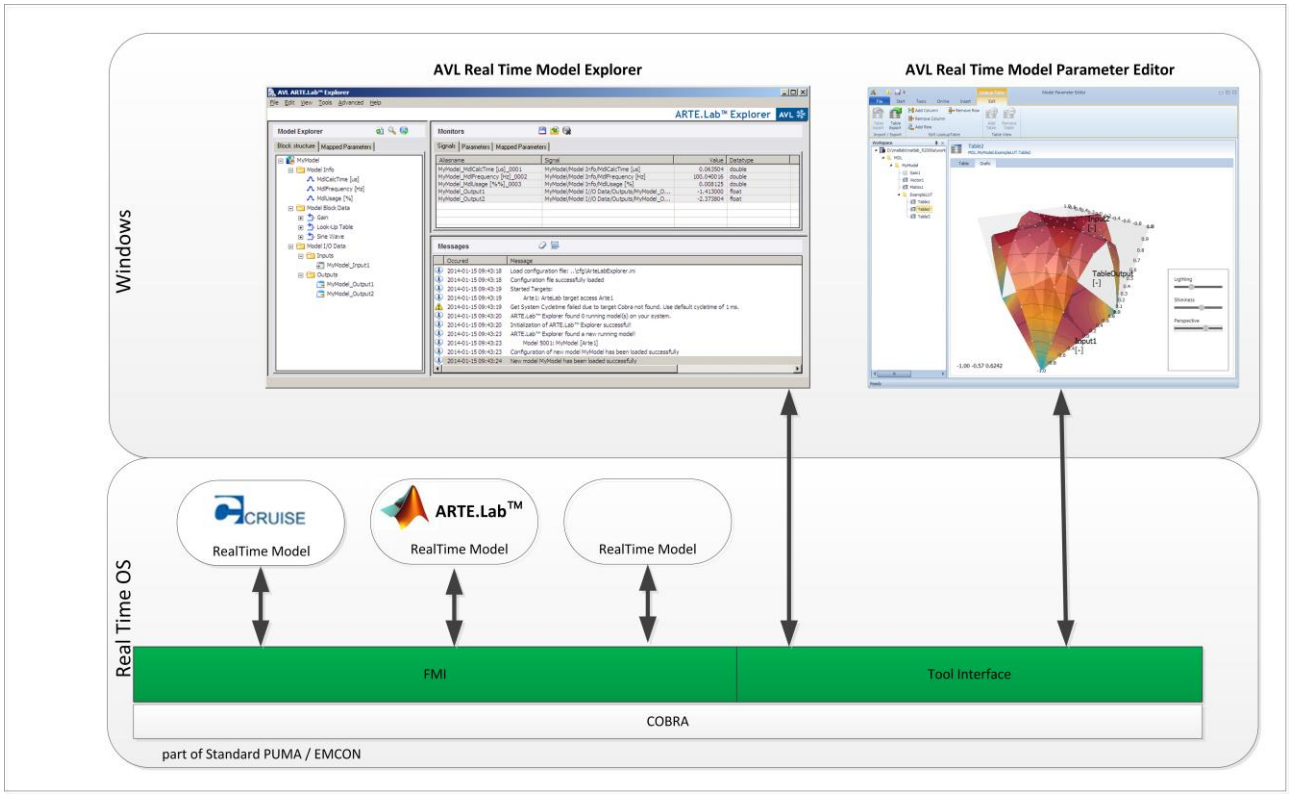

Figure 5-4 Using FMI to enable co-simulation of simulation models coming from different simulation authoring tools

Link to internal working documents:	

6 Prototype implementation for CRYSTAL_TI_6.13.1_1

As described by TI_6.13.1_1 (see Section 2.3.1), the integration of simulation models is an essential task to extend the AVL Santorin tool to a simulation model data backbone. Consequently, development efforts are started with this integration in mind. More specifically, **AVL Santorin** was extended to support simulation models defined by the AVL vehicle simulation tool **AVL CRUISE**. AVL CRUISE tool itself is also defined as a CRYSTAL brick and is described in WP6.3.

This integration would then enable further interoperability aspects such as linking simulation models and measurement results produced by simulation model execution in one of the testing phases described in WP3.4.

This chapter describes the current prototype status about the integration of AVL CRUISE simulation models in AVL Santorin to enable the latter tool to act as simulation model data backbone. For the prototype integration standard interoperability mechanism such as SOAP and CORBA are applied. OSLC is not yet considered here but is planned to be built on top of the current approach in order to link simulation models stored in the data backbone to other data categories (e.g. requirements). In that sense, this integration is considered to found the basis for such an OSLC extension, which will be considered in the next phases of the project and will potentially generalize the presented prototype approach toward standardization aspects such as provided by OSLC.

6.1 Architecture and data structure

6.1.1 Architecture

Due to the prototype implementation activities, AVL CRUISE models can be now stored in AVL Santorin (which is based on an ORACLE DB with an ASAM ODS compliant data model layer). Further architectural components are ModelParam, AVL Simulation Desktop and AVL Navigator. In Figure 6-1, this architecture is illustrated.

The component ModelParam acts a service provider for data management for simulation models and is responsible for communicating and transferring of model data to Santorin (currently done via a native CORBA interface, which is actually provided by AVL Santorin).

AVL Simulation Desktop and Navigator are clients of the service provider ModelParam. While AVL Simulation Desktop is the frontend for the simulation model itself (including AVL CRUISE models), AVL Navigator is used for high level navigation over various data categories stored in AVL Santorin (including now simulation models) and their basic manipulation (e.g. copying, searching and previewing).

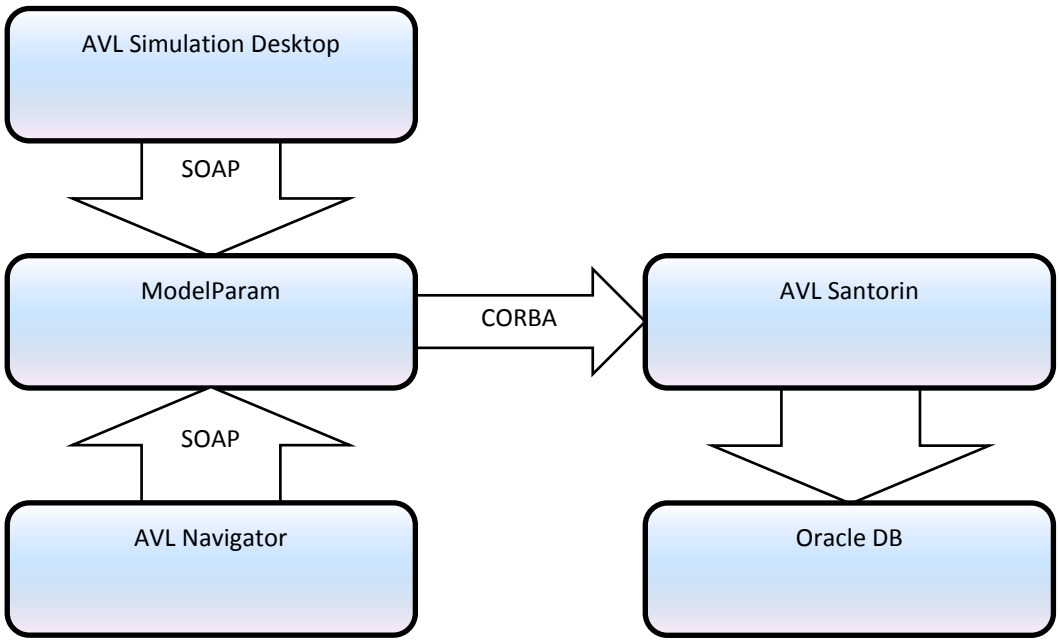

Figure 6-1: Architecture of the simulation model integration in AVL Santorin

For that reason ModelParam exposes a series of web services used by Simulation Desktop and Navigator to manipulate Simulation Desktop projects and their parts:

- *ModelParamWS/ParameterSetTaskService:*
  Provides operations for creating, retrieving and updating parameter sets.
- *ModelParamWS/ResourceTransferService:*
  Used for storing and retrieving raw data attachments to blocks.
- *ModelParamWS/LockingService:*
  Provides locking semantics for parameter sets and library elements.
- *ModelParamWS/NavigationExService:*
  Provides copy/move/rename/delete and other navigation operations for folders, parameter sets and library elements.
- *ModelParamWS/LibraryElementTaskService:*
  Provides methods for creating, retrieving and updating library elements.
- *OneParamV11_NavigationWS/Service:*
  Low-level/legacy interface to the navigation functionality required by AVL Navigator.

6.1.2 Data structure

In order to deal with AVL Cruise models, the ModelParam component is extended by the following data structure: An AVL Simulation Desktop project is stored in ModelParam as a parameter set of type MBBParameterSet. MBBParameterSet can contain following blocks.

In order to deal with AVL Cruise models, the ModelParam component is extended by the following data structure: An AVL Simulation Desktop project is stored in ModelParam as a parameter set of type MBBParameterSet. MBBParameterSet can contain following blocks:

- *Model (MDL):*
  Contains essential model data including model elements (e.g. Front Brake and Friction Clutch in Figure 8-2), their properties, connections etc. Multiplicity 0-n.
- *Sharing (SHR):*
  Contains information which elements are shared between which models (MDL blocks) allowing successful merging of such model parts when the parameter set is opened in AVL Simulation Desktop. Multiplicity 0-1.
- *Element (ELM):*
  Contains data about a standalone element i.e. a part of a model explicitly selected by the user to be represented as a separate entity in the database. Multiplicity 0-n.
- *Data pool (DPL)*
  Contains AVL Simulation Desktop data pool i.e. a collection of property objects (e.g. Mass Properties in Figure 8-5) which can be reused in all compatible locations in models (MDL blocks). Multiplicity 0-1.
- *Parameters (PRM)*
  Contains AVL Simulation Desktop parameters i.e. a collection of named values which can be referenced by name in compatible locations in models (MDL blocks). Parameters offer an additional level of indirection and can stand in for simple values as well as more complex data like data pool elements and model elements. Multiplicity 0-1.
- *Scenarios (SCN)*
  Contains AVL Simulation Desktop scenarios i.e. a collection of named sets of values which can be referenced by name in compatible locations in models (MDL blocks). Scenarios are a generalization of the concept of parameters offering a way to simultaneously control several values of possibly different types. Multiplicity 0-1.
- *Cases (CAS)*

Contains AVL Simulation Desktop cases organized in different case sets. Cases allow parameter and scenario variation over different simulation runs. Multiplicity 0-1.

- *Embedded file (EMF)*
  Contains a file embedded into an AVL Simulation Desktop project. Multiplicity 0-n.

Only some attributes of entities represented by blocks are modelled explicitly in ModelParam and most of them can be read in AVL Navigator as so-called “custom attributes”. Other attributes which make up the bulk of the data are stored as XML attachments to blocks. Such attachments can be interpreted only by AVL Simulation Desktop.

6.2 Performed activities

6.2.1 Architectural design

The foundations of the architecture have been defined by members of OneParam and Simulation Desktop Framework teams in corresponding workshops. Several options have been considered, ranging from full modeling of AVL Simulation Desktop data model in ModelParam to no modeling at all i.e. storing AVL Simulation Desktop projects as blobs attached to otherwise-empty blocks. The selected option lies between those two extremes – projects are split into overlays and each overlay is stored as a separate block with only some attributes modeled in ModelParam.

6.2.2 The program *wsdlpy*

In order to use SOAP-based web services from Python we developed the program *wsdlpy*[1] which parses WSDL and generates Python proxy classes for services and their request and response data structures. XSD parsing and code generation is delegated to PyXB (http://pyxb.sourceforge.net/).

6.2.3 ModelParam core implementation

The initial set of ModelParam web services necessary for basic folder, parameter set and block navigation and manipulation has been implemented based on OneParam. It turned out that a new, simpler and cleaner interface for navigation was required for efficient use in AVL Simulation Desktop. The new interface has been designed and implemented in a separate web service (ModelParamWS/NavigationExService).

6.2.4 Overlays

To support partial modeling of AVL Simulation Desktop project structure in ModelParam, splitting of projects into overlays has been implemented. At the moment only some project data can be represented with overlays (models, sharing-between-models information and standalone elements) but those overlays carry the most important information and have the most complex structure with many special cases (e.g. element sharing and model embedding).

6.2.5 Preview in AVL Navigator

To help the user choose the right model directly in AVL Navigator a preview plugin for AVL Navigator has been implemented. At the moment preview for MDL blocks shows the model preview image as seen e.g. on Info page of the Project category in AVL Simulation Desktop.

6.2.6 Element and model library

To provide the possibility to reuse models and explicitly selected parts of models (standalone elements) the OneParam concept of library handling has been integrated into ModelParam and Simulation Desktop. Once a so-called MDL or ELM block is created in a library project, typically by copying it from a regular project in AVL Navigator, it can be copied or, even more important, linked into a parameter set representing an AVL Simulation Desktop project. In case of linking, a copy of the block is inserted into the parameter set but the link to the original is maintained. When the original is later modified the user can choose to update all or only some linked copies to the latest version.

[1] *wsdlpy* is a program that takes WSDL (XML description of a web service) as input and generates Python code needed to access the service from Python.

6.2.7 Custom attributes

To make the presentation of ModelParam blocks in AVL Navigator more informative the concept of custom attributes has been introduced into OneParam and used in ModelParam. Values of such attributes are shown in table view of the parameter set contents in AVL Navigator.

6.3 Functionality in the current prototype

The current prototype can be used to:

- Browse the ModelParam folder hierarchy and preview information about stored projects without opening them using AVL Simulation Desktop.
- Store projects into a database and load them back, using AVL Simulation Desktop. Data pool, parameters, scenarios, cases and embedded files are not yet supported.
- Create empty parameter sets (projects) using AVL Navigator.
- Manipulate (copy or move between folders, rename, delete) parameter sets (projects) using AVL Navigator.
- Manipulate (copy or move between projects, rename, delete) blocks (models, sharing information and standalone elements) using AVL Navigator.
- Store models and standalone elements into a library project using AVL Navigator.
- Copy or link models and standalone elements from a library project into parameter sets (projects) using AVL Navigator.
- Update outdated links from library to parameter sets.

7 Terms, Abbreviations and Definitions

Please add additional terms, abbreviations and definitions for your deliverable.

CRYSTAL	**CR**itical S**YST**em Engineering **A**cce**L**eration
R	Report
P	Prototype
D	Demonstrator
O	Other
PU	Public
PP	Restricted to other program participants (including the JU).
RE	Restricted to a group specified by the consortium (including the JU).
CO	Confidential, only for members of the consortium (including the JU).
WP	Work Package
SP	Subproject
ALM	Application Lifecycle Management
PLM	Product Lifecycle Management
ASCII	American Standard Code for Information Interchange
ASAM	Association for Standardization of Automation and Measuring Systems
ASAM ODS	ASAM Open Data Services
ECU	Electronic Control Unit
EA	Enterprise Architect
FMI	Functional Mock-up Interface
HiL	Hardware-in-the-Loop
IOS	InterOperability Specification
ISO 26262	ISO norm for “Road vehicles – Functional safety”
OSLC	Open Services for Lifecycle Collaboration
OSLC AM	OSLC Architecture Management Specification
OSLC RM	OSLC Requirements Management Specification
OSLC QM	OSLC Quality Management Specification
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RTE	Runtime Environment
RS-232	Serial interface for data transfer
SiL	Software-in-the-Loop
SL	Simulink
SysML	System Modelling Language
TCP/IP	Transfer Control Protocol / Internet Protocol
TeSaCo	Technical Safety Concept

UC	Use Case
UML	Unified Modelling Language
UUT	Unit Under Test

Table 7-1: Terms, Abbreviations and Definitions

8 Annex

8.1 Annex I: Video

The functionality listed in Chapter 6 is illustrated by video under the following link:

https://projects.avl.com/11/0154/Data%20Exchange/007_Work/SP6_RnT_Activities/6.13%20Simulation%20Models/Deliverables/CRYSTAL_D_613_021_CRUISE_SANTORIN_Integration.mkv

8.2 Annex II: Screenshots

Some functionality listed in Chapter 6 is illustrated by the following screenshots:

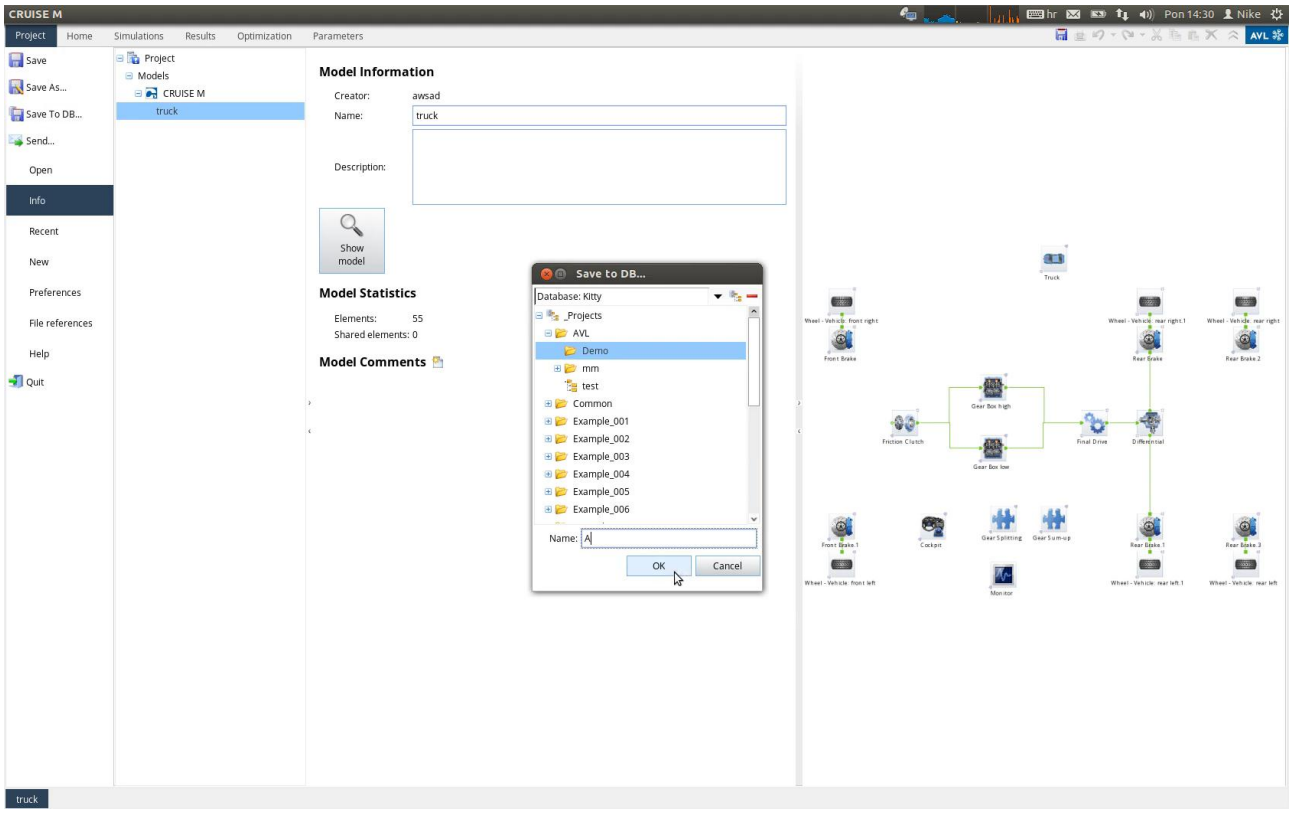

Figure 8-1: Saving a project to a database using AVL Simulation Desktop

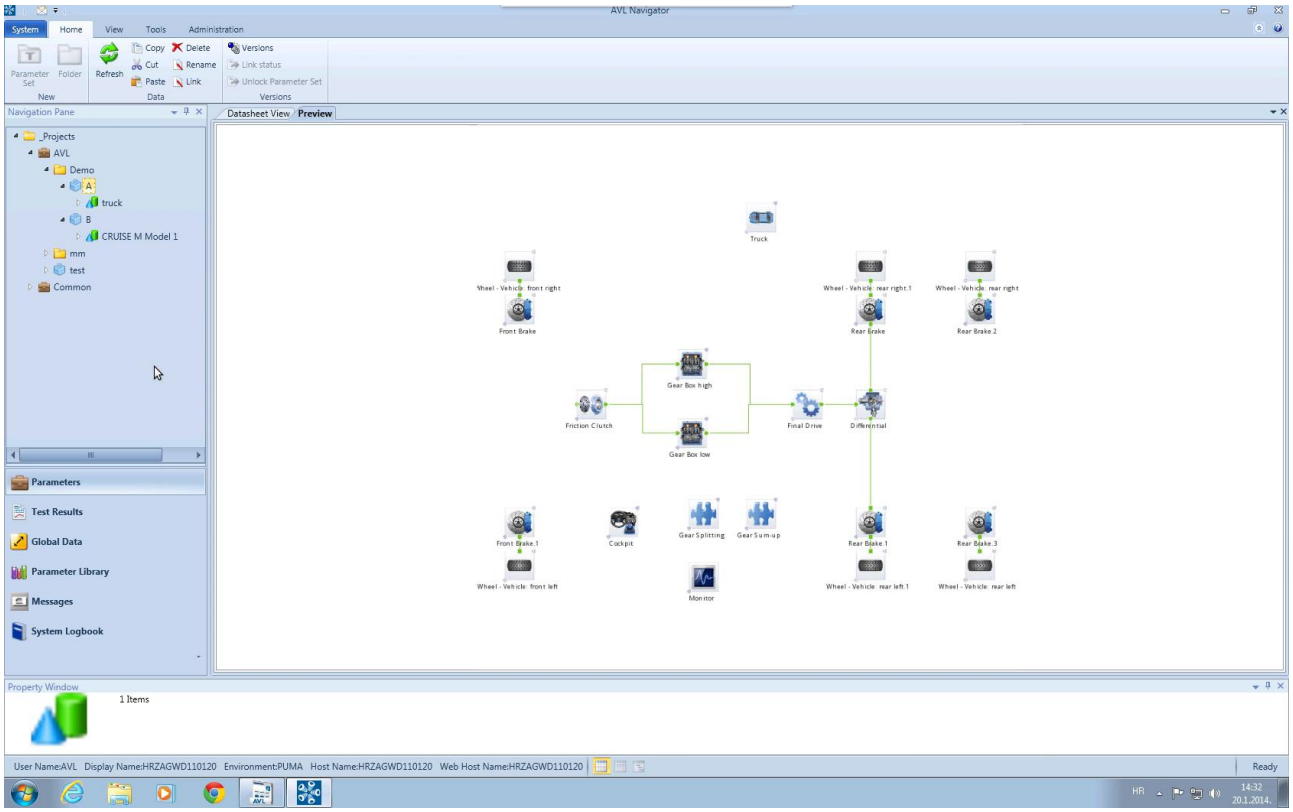

Figure 8-2: Previewing a model using AVL Navigator

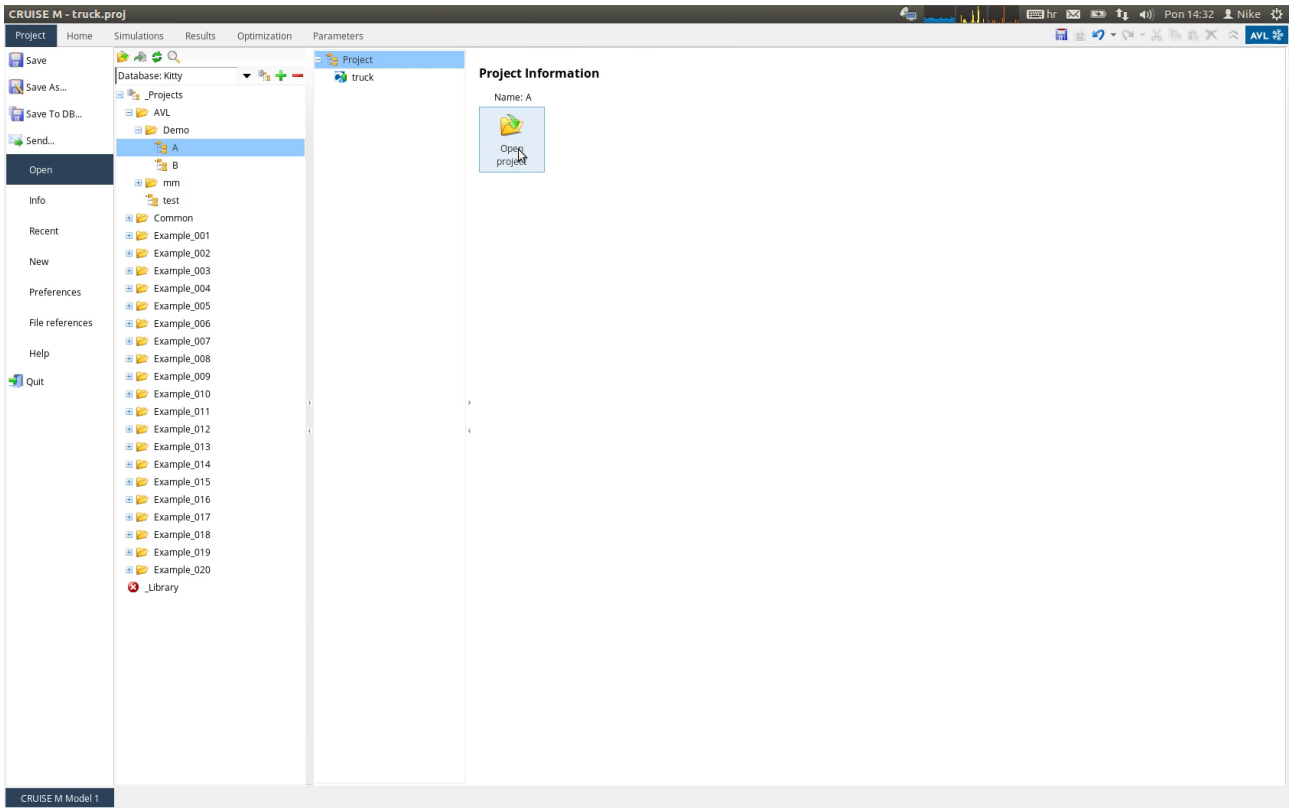

Figure 8-3: Opening a project from a database using Simulation Desktop

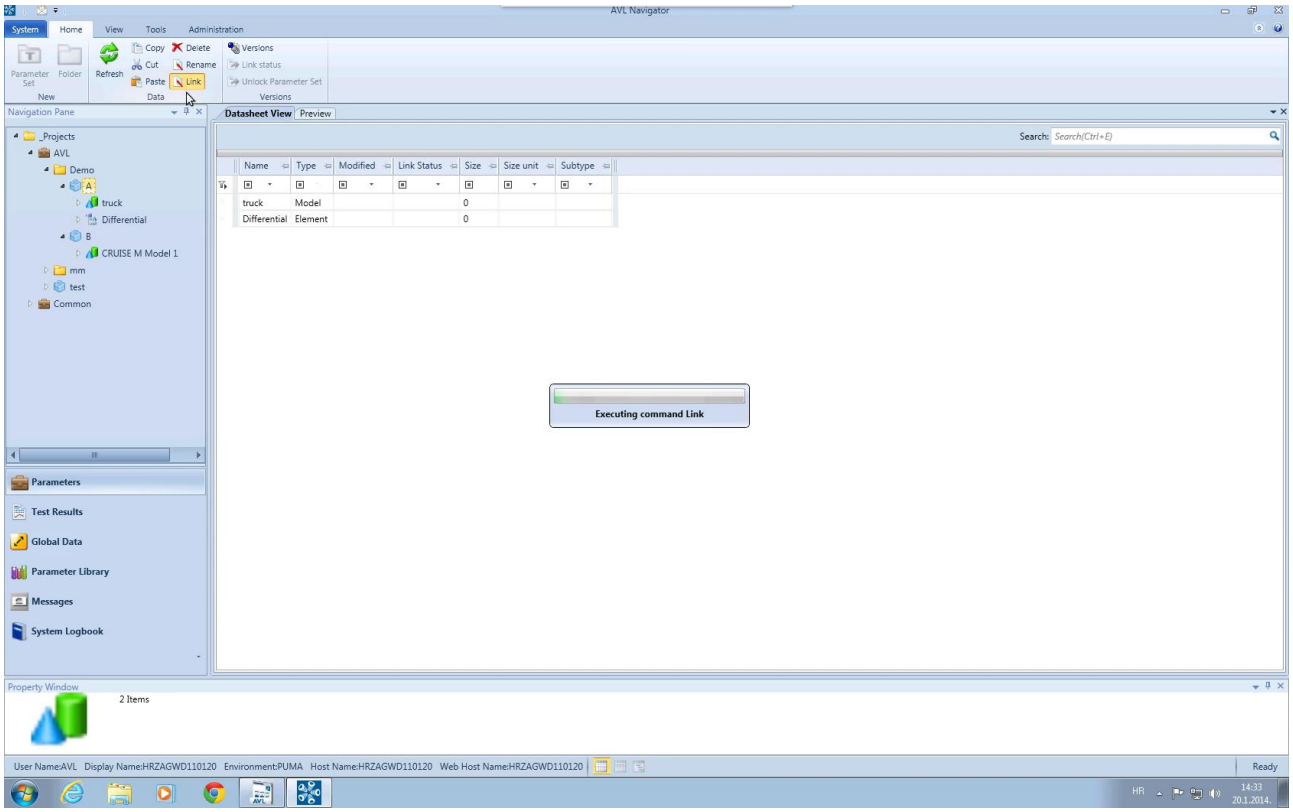

Figure 8-4: Linking a standalone element from library into a project using AVL Navigator

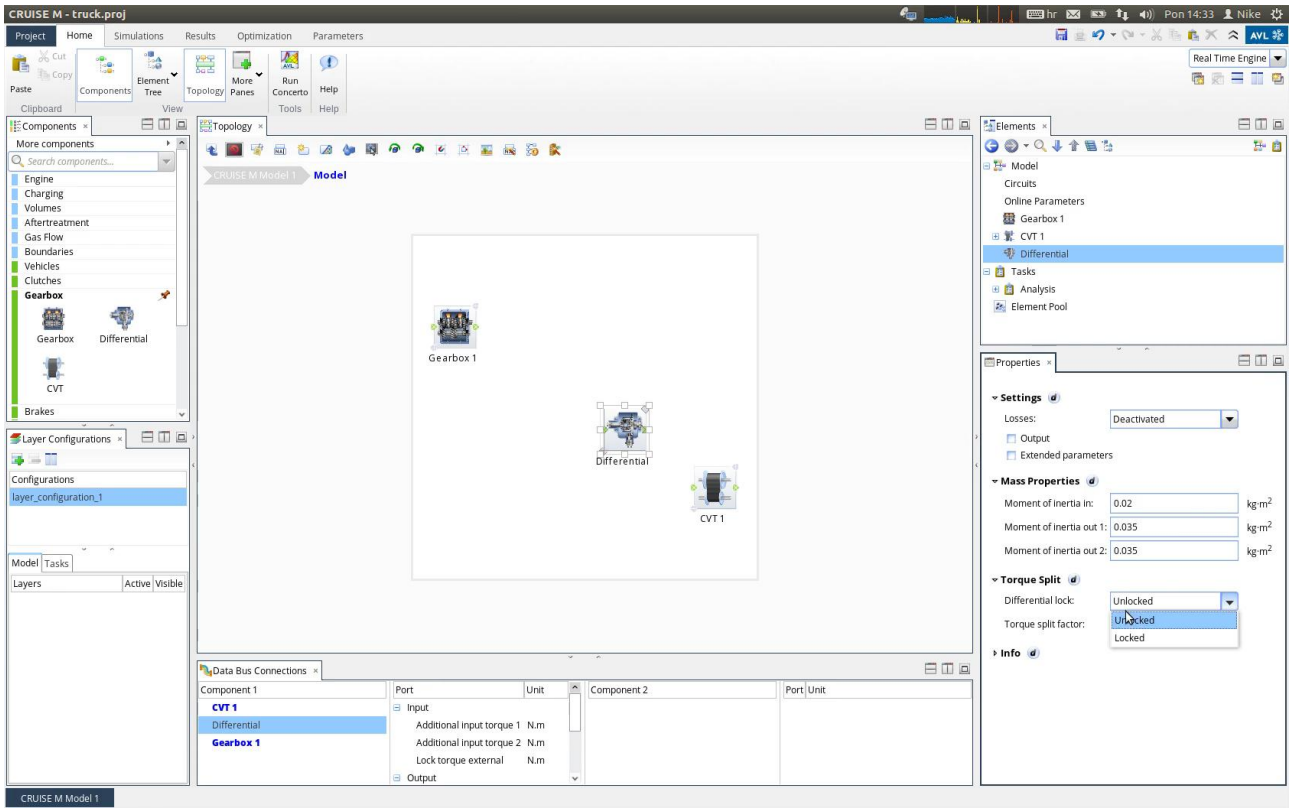

Figure 8-5: Modifying a standalone element using AVL Simulation Desktop

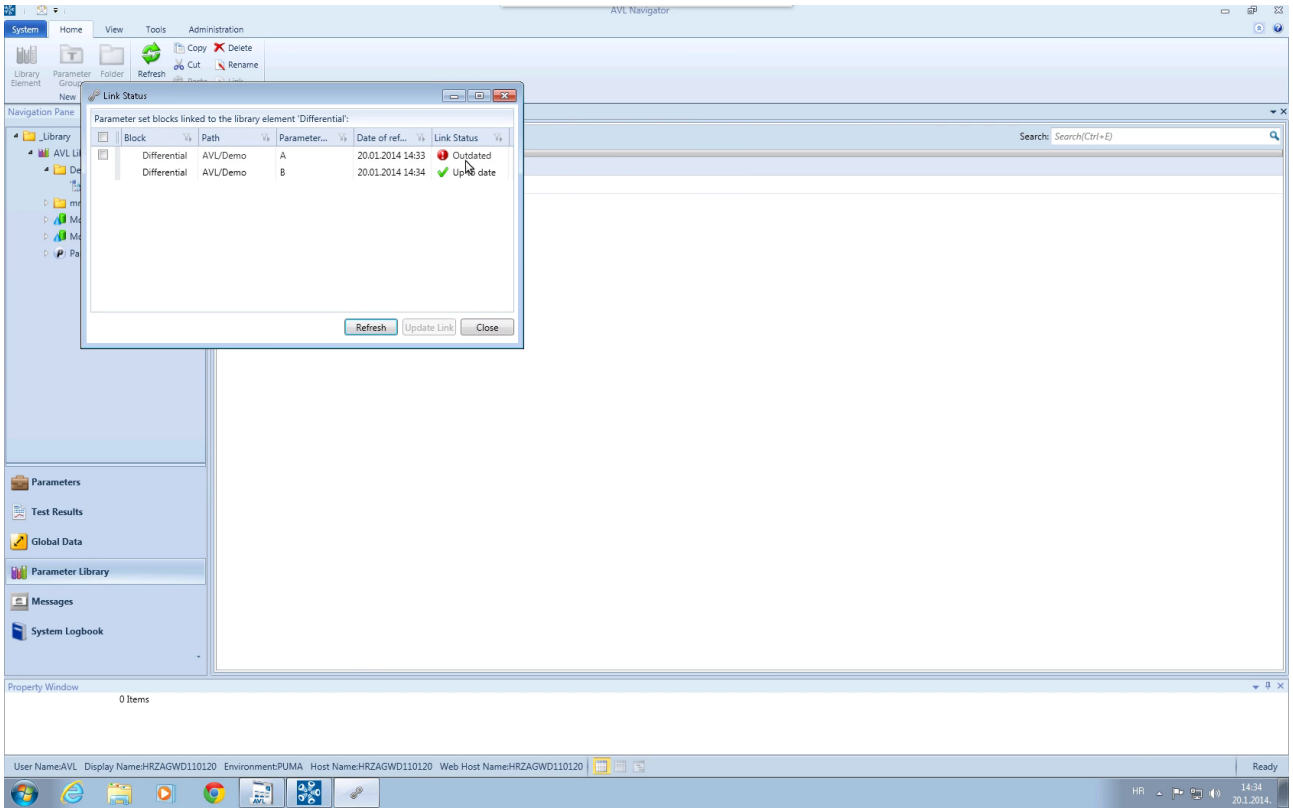

Figure 8-6: Updating an outdated link in AVL Navigator