 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

CRYSTAL Variability Management V1

D610.011

DOCUMENT INFORMATION

Project	CRYSTAL
Grant Agreement No.	ARTEMIS-2012-1-332830
Deliverable Title	CRYSTAL Variability Management V1
Deliverable No.	D610.011
Dissemination Level	CO
Nature	R
Document Version	V1.0
Date	2014-02-28
Contact	Jason Mansell
Organization	TECNALIA
Phone	(+34)664 10 45 97
E-Mail	jason.mansell@tecnalia.com

AUTHORS TABLE

Name	Company	E-Mail
Jason Mansell	TECNALIA	jason.mansell@tecnalia.com
Carlos Zubieta	ORBITAL	carlos.zubieta@orbital-aerospace.com
Martin Becker	Fraunhofer IESE	Martin.Becker@iese.fraunhofer.de
Konstantin Keutner	Siemens	konstantin.keutner@siemens.com
Andrea Leitner	VIF	Andrea.leitner@v2c2.at
Gerald Stieglbauer	AVL	Gerald.stieglbauer@avl.com
Adeline Silva Schäfer	Fraunhofer IESE	Adeline.Schaefer@iese.fraunhofer.de
Arjan Mooij	TNO	arjan.mooij@tno.nl
Mark Van Den Brand	Technical University Eindhoven	m.g.j.v.d.brand@tue.nl
Rob Albers	Philips Medical Systems Nederland B.V.	r.albers@philips.com

CHANGE HISTORY

Version	Date	Reason for Change	Pages Affected
V0.1	25/10/2013	Included information from Orbital	7-9
V0.2	29/11/2013	Included Contribution from Fraunhofer, Siemens.	10-12
V0.3	15/01/2014	Consolidated version with final contributions from Frauhofer, Orbital, Siemens, TNO, Tu/E, Philips, AVL, VIF, TECNALIA.	All

CONTENT

Content of Figures

Content of Tables

1 Introduction

1.1 Role of deliverable

Within CRYSTAL one of the key step for success is the maximization of the reuse of existing knowledge in the creation of new critical systems in a faster, secure, certified and reliable manner. In this direction this deliverable sets the basis for the provision of variability management practices, methods and tools for the industrial cases within CRYSTAL.

This deliverable provides the specification of WP6.10 Bricks that involve tool support and specific tool enhancement that will be used in the use cases.

1.2 Relationship to other CRYSTAL Documents

This document is the first in a series of three reports:

- CRYSTAL_D610_011 – Crystal Variability Management - V1 (this document)
- CRYSTAL_D610_012 – Crystal Variability Management – V2
- CRYSTAL_D610_013 – Crystal Variability Management – V3

Use Cases cited in this deliverable:

- CRYSTAL_D203_011 - Use Case 2.3 “Mission Support Equipment” (EADS-CAS),
- CRYSTAL_D304_011 - Use Case 3.4 “Test Case Definition” (AVL)
- CRYSTAL_D401_010 - WP4.1 (Philips HealthCare)

1.3 Structure of this document

The remainder of this document is composed of 6 sections detailing the current status and plans for the WP6.10 technical bricks development. Each section is structured in the following way:

- Description: provides a description of what the technical basis of the brick is and how the Brick can be used
- Improvements in CRYSTAL: describes the intended further development within CRYSTAL and the plans to use it within the CRYSTAL use cases.
- Integration within IOS: Describes the plans for integration with the IOS.

Task	Company	Brick Name
6.10.3	Fh IESE	Variant Analysis
6.10.5	AVL	iGeM
6.10.6	AVL	TFMS
6.10.7	AVL	Creta/Cameo
6.10.8	Siemens	DSL
6.10.9	Orbital	AUGE

Table 1-1: Bricks x Companies

This document will be a living document that in its second release will detail the technical work done as well as the result of the Use Case validation.

2 Variant Analysis Brick (Task 6.10.3)

2.1 Description

Name:	Variant Analysis
Contact:	Slawomir.Duszynski@iese.fraunhofer.de
Technical Information: This brick will be a standalone software application with graphical HMI allowing the user to simultaneously analyse the similarity of a group of up to 32 software artefact variants (especially source code, documents). The application determines and visualizes the detailed information on similarity of the artefacts variants on all levels of abstraction.	
Operation System/platform	Windows/Linux
Version	1.0
Type of Input Data	Software Artefacts (source code, text)
Type of Output Data	Similarity graphs, matrices and tables
Dependencies	Eclipse
License	Terms and conditions agreed in CRYSTAL APCA
Additional information	N/A

2.1.1 Tool/method description: what will you find in Variant Analysis

Variant Analysis is an approach and tool to identify commonality and variability in the engineering artefacts (for instance, source code) of existing system variants in an efficient and effective way. It compares several artefact variants in parallel and supports an interactive commonality-variability-analysis on multiple levels of abstraction. In this way, it enables the user to assess the reuse potential in existing artefacts and decide on the optimal strategy for introducing variation management approaches.

Variant Analysis is intended to be used in the situation when variation management approaches should be used retroactively, after the subject software assets are already developed. In many practical situations, the need for asset customization is often not visible upfront, but rather emerges during asset lifetime. In such a case, the variants are often created in ad-hoc manner – for example by cloning the original artefact and changing it according to the specific requirements of the customer. Subsequently, the artefact variants are maintained in parallel independently of each other. The described approach may be a viable and easy-to-realize short-term solution, but in a longer time it causes serious maintenance problems. These problems can be alleviated by adoption of a suitable variation management approach. Therefore, the existing assets need to be analysed and transformed into a new, generic and reusable form. Variant Analysis delivers the similarity information for these activities. It can also be used for any other purpose where detailed and accurate information on the similarity of multiple artefact variants is needed.

Variant Analysis uses a hierarchical set similarity model which enables very efficient similarity analysis and supports easily understandable result presentation. Compared to other similarity analysis approaches, Variant Analysis is particularly efficient for large software systems and a high number of analysed software variants.

2.1.1.1 Installing / Using method Variant Analysis

Variant Analysis is an Eclipse-based application distributed with the use of the Eclipse Update Site mechanism. In order to use the tool, the user needs to have the Eclipse environment installed on the computer. Depending on the environment configuration, the installation process of Variant Analysis can also include an installation of Eclipse-based open-source model management frameworks used by the tool (e.g. Eclipse Modelling Framework).

The tool binaries are accompanied by documentation (PDF) explaining tool use and result interpretation. The tool license is displayed before the installation process starts.

To use the tool, Eclipse needs to be started. The specific Eclipse perspective, consisting of Variant Analysis views, will be displayed in the Eclipse environment. The usage of the tool consists of three main steps:

- Artefact import: the specific artefact variants which need to be analysed for similarity should be added to the analysis project using the fact extraction wizard. In the wizard, the user has the possibility to filter the imported artefacts (e.g. by specifying the accepted file extensions) and configure other extraction options (e.g. concerning the internal structure of the artefact).
- Artefact similarity analysis: in the analysis wizard, the user can specify the options for similarity analysis. The options include, among others, the choice of a similarity analysis algorithm and the choice of a similarity evaluation function.
- Result browsing and interpretation: the structure hierarchy of the analysed artefact variants is displayed in a diagram. For each artefact or its constituent element, similarity information relating it to other variants of the selected element is displayed. The user can navigate the structure of the artefact and receive information related to elements on any level of abstraction (the “details on demand” principle). The information bases on the hierarchical set data model and is provided in the form of diagrams, matrices and tables. It can be exported to output files for further processing in other tools.

2.1.1.2 TEST Variant Analysis installation / Example of usage of the Method

Initially, the installation can be verified by the means of Eclipse installation details dialog. Eclipse should report the feature “Fraunhofer Variant Analysis” and the corresponding plugins such as “de.fhg.iese.save.structure.an.variantanalysis” as installed. If the installation is successful, the SAVE Eclipse perspective should be shown by default, and the user should be able to enable the two related views: “Variant Analysis Result View” and “Variant Analysis Query View”.

2.2 Improvements in CRYSTAL

The tool delivers information for reengineering activities aiming at developing generic, reusable artefacts which are equivalent to the multiple input artefact variants. In the basic usage, the use of the tool for source code should be demonstrated. Advanced tool improvements include the usage for other textual artefacts, such as requirements, test cases and specification documents, as well as for design models. For this purpose, suitable analysis algorithms and equivalence functions need to be defined.

2.2.1 Case Study coverage

Variant Analysis will be applied in the Use Case 2.3 “Mission Support Equipment” in the Aerospace domain, led by EADS Cassidian. The Use Case focus is on integration of the bricks related to requirement engineering, artefact traceability, safety analyses and variability management. The Variant Analysis tool will support the user in analysis and reengineering of input artefacts, e.g. textual requirements and specifications, in order to introduce suitable variability management practices to these artefacts.

2.3 Integration within the IOS

The implementation of an IOS data connector, supporting data exchange in both directions (importing the artefact data and exporting the similarity results), is envisioned**.**

3 AVL iGeM Brick (Task 6.10.5)

3.1 Description

AVL iGEM is a tool which is used in the context of engine and vehicle test beds. These test beds are used for early validations, e.g. of the engine while the rest of the vehicle is still simulated. Important aspects especially for engines are emission tests, because vehicles have to adhere to certain limits.

The AVL iGEM product line guarantees the correct implementation of the latest legislative code in the emission automation for engine and vehicle testbeds selecting from a huge range of variants. The modular structure of the application allows selecting a set of test applications in advance but also an upgrade or extension later on. iGEM offers high scalability and also allows simple adjustments for different testbed configurations to be made based upon individual user needs.

The GEM Engine product line covers Engine Emission Research & Development and Certification testing for Heavy Duty On-Highway, all Off-Highway and certification-like the testing of Light Duty and passenger car engines on dynamic engine testbeds.

Frequent changes in the worldwide rulemaking of emission legislation increase the need to keep up to date with emission automation. New requirements like EPA 40 CFR Part 1065 and Off-Highway regulations as well as Euro 5/6 are a challenge to modern emission testing. AVL iGEM Engine and GEM301 EC guarantee a correct implementation of the latest legislative code in the emission automation for engine test beds.

The product consists of a base module, which is the interface to the engine automation PUMA Open. The application covers ECE, EPA, Japanese, Chinese, ISO and World Harmonized emission regulations and legislation for certification as well as R&D purpose. The legislative tests can be executed in raw/partial flow as well as diluted/full flow mode. iGEM Engine as well as GEM301 EC supports the fuel types Diesel, Gasoline, Biodiesel, Methanol, Ethanol, Propane, Butane, CNG and LPG for all applications. All tests from the above listed legislations are already pre-parameterized for device & engine control, data storage and calculation & reporting according to legislation.

The application is adaptable to the configuration of the test cell and requirements of the customer. The modular structure of the application allows selecting a set of test applications in advance but also an upgrade or extension later on. Customer specific applications and extensions to the standard modules can be easily implemented.

AVL iGEM Vehicle provides ready-to-use application packages for passenger car, medium & heavy duty truck and motorcycle.

iGEM Vehicle is used for certification, research & development, or for conformity of production purposes. iGEM Vehicle offers the best solution for passenger car, medium and heavy duty truck, and motorcycle emission automation. It also provides varied and personalized application packages.

AVL iGEM Offline test data evaluation is an innovative solution for efficient data analysis of exhaust emission tests according to legislative demands.

iGEM Offline includes a series of effective tools and offers the possibility for authorized users to change or expand an existing record configuration. The Formula Editor helps to change calculation variables and formulas and add them into the database. The configuration can be adapted to comply with new legislation or modified technical conditions. Report templates can be created and modified easily via drag and drop operations in the Report Layout Editor. Several different types of reports can be created besides the typical standard reports such as online and modal reports;

specimen, equipment and consumable data record sheets; statistics COP and audit reports; testing series reports and also combinations of different types of reports.

3.2 Improvements in CRYSTAL

Major tool improvements will be about an improved interoperability with other tools included in the overall activities of testing a vehicle at different vehicle development phases. A special focus of collaboration could be the interoperability with the brick Simulation Model Backbone Database (B3.83), which will be developed in WP6.13.

3.2.1 Case Study coverage

According to CRYSTAL's application document, the AVL iGem (Brick 3.50) is associated with WP3.4 (UC3.4a). At the current stage of UC definition, iGEM is not yet explicitly included, but could 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. This specification could be transferred to a corresponding iGEM configuration during the improvement of the UC specification.

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

3.3 Integration within the IOS

The main objective of this task is to improve the interoperability with other tools by applying IOS concepts in WP3.4. Since the role of AVL iGem 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.

4 AVL TFMS Brick (Task 6.10.6)

4.1 Description

AVL TestFactory Management SuiteTM (TFMS) is a comprehensive system for the standardization and automation of the core processes in the field of testing for the automotive domain. Test bed systems provide a high degree on variability, because various units can be tested in various environments. The main task of AVLS TFMS is the efficient management of all data related to test orders, test equipment, and units under test.
AVL's approach is to integrate and to interact with the test automation system and other existing business systems such as project management systems, unit under test data bases and calibration data management systems. Unused productivity potential of a test facility will be available.
While there is a trend towards increasing complexity of the processes and diversity of the systems in test facilities, the test costs are to be reduced drastically. Simultaneously, the reliability, reproducibility and quality of the process are to be improved. The AVL TestFactory Management Suite™ - TFMS supports the users in achieving these seemingly contradictory objectives.
In order to support the seamless implementation of test processes, the system controls the work steps and provides all data and documents that are relevant in the corresponding step. This way, the planning, definition and implementation of test jobs as well as inventorying, maintenance, and calibration of test equipment are managed and optimized with the help of the AVL TestFactory Management Suite™. The existing system environment within the test facility is not replaced, but the TestFactory Management Suite™ integrates and interacts with the existing systems, such as project management, unit under test and calibration data management.
Thus, the AVL TestFactory Management Suite™ taps into the unused productivity potential of a test facility through cross-networking and providing the corresponding information at the right time and place.

4.2 Improvements in CRYSTAL

The major objective of this task is to improve the interoperability with other tools by the application of IOS concepts. A special focus of collaboration could be the interoperability with the brick Simulation Model Backbone Database (B3.83), which will be developed in WP6.13.

4.2.1 Case Study coverage

According to CRYSTAL's application document, the AVL TFMS (Brick 3.48) is associated with WP3.4 (UC3.4a). At the current stage of UC definition, TFMS is not yet explicitly included, but could play a role at a later use case definition phase.

A critical issue in WP3.4 is that during the overall process of developing a vehicle various testing scenarios with different test bed configurations have to be set up. Managing all these different variants and data re-use across development phases is one of the greatest challenges of this WP. The explicit role of TFMS and variant management especially in term of interoperability challenges has to be defined by WP3.4 by future enhancements.

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

4.3 Integration within the IOS

The main objective of this task is to improve the interoperability with other tools by applying IOS concepts in WP3.4. Since the role of AVL TFMS 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 AVL Creta/Cameo (Task 6.10.7)

5.1 Description

Name:	AVL Creta/Cameo
Contact:	Andrea.leitner@v2c2.at
Technical Information	
Operation System/platform	Windows
Version	
Type of Input Data	Calibration Set-up, Measurement Results, Initial Calibration Data, Test Cases
Type of Output Data	Calibration Model, Calibration Data
Dependencies	Testbed-Set up, UUT parameters, AVL Cruise/Boost brick, AVL Data Backbone
License	
Additional information	

5.1.1 Tool/method description: what will you find in [name of brick]

As a central calibration data management system for xCU parameters, AVL CRETA™ allows the central storage, conflict-free merging, and traceable documentation of calibration datasets and variants during series calibration projects. Due to the high amount of calibration labels, more and more vehicle variants, globally distributed team assignments and close collaboration with partners, calibration data management becomes highly complex.
Today's complex calibration projects including multiple xCU's with more than 40.000 parameters/labels and this large number of calibration variants require standardized and easily applicable processes to maintain a high level of quality. AVL CRETA™ ensures that every development engineer follows your company standards.
Submitting calibrations, creating a report, checking the project status, getting information on dataset history, or the comparison of calibration results are one click tasks. This saves a substantial amount of time for engineers allowing them to focus on their real work instead of sending and documenting files and datasets.

AVL CAMEO Powertrain Calibration Environment is a powerful tool that gives you one window onto handling the complete calibration process. It performs the Modelling and optimization task based on the measured engine responses. This means that AVL Cameo is a tool to optimize calibration datasets and is therefore closely related to AVL CRETA.

We consider AVL CAMEO and AVL CRETA as one tool suite for calibration here.

5.2 Improvements in CRYSTAL

Major tool improvements will be about an improved interoperability with other tools included in the overall activities of testing a vehicle at different vehicle development phases. A special focus of

collaboration could be the interoperability with the brick Simulation Model Backbone Database (B3.83), which will be developed in WP6.13. In addition, improving the integration with simulation tools and testbed environments will be another key focus.

5.2.1 Case Study coverage

Both tools, AVL Creta and AVL Cameo are strongly are strongly related to calibration processes in various vehicle testing scenarios. The brick thus depend on the AVL use case of WP3.4.

In this WP the test case conceptualization for testing a vehicle is described. A general pattern of a test environment is about UUT calibration as shown in Figure 5-1.

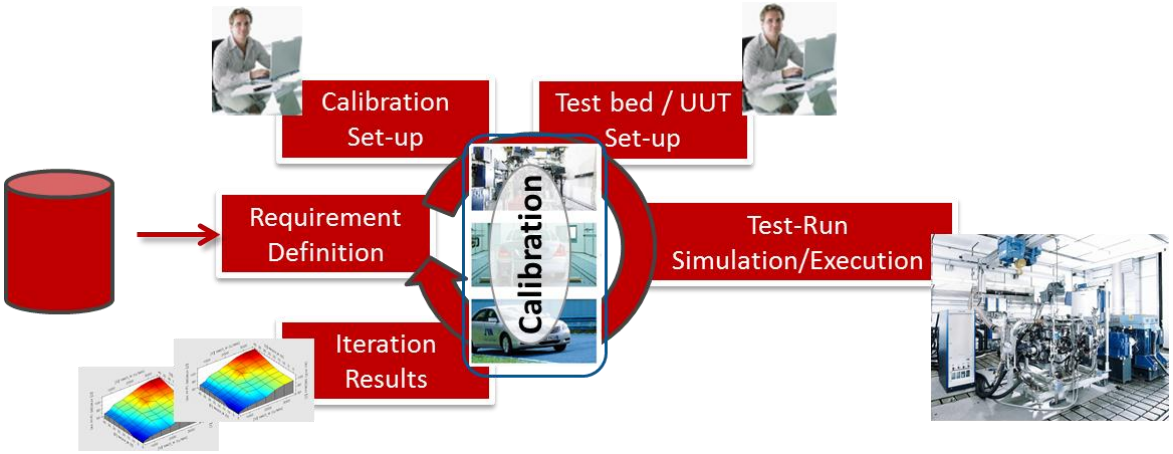

Figure 5-1: Test and calibration iteration pattern

The *test case conceptualization* is divided by two set-ups (*test modules*): the calibration and the test bed set-up. The test bed set-up configures the testing environment (with further sub-modules such as needed measurement devices, UUT 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 theses set-ups with all there sub-modules finally leads to the implementation and integration of the overall test set-up, which can be executed with a set of given test-runs.

In many cases, the test modules *test case execution* and *calibration* are clearly separated: During a test-run execution measurement results are associated to given test run input vectors. These pairs of date are used in corresponding calibration tools to create a calibration model that interpolates even not tested constellations and supports in speeding up the calibration process as a whole.

If the test case execution is entirely done by simulation in early development cycles, simulation results based on an initial simulation model parameter set can be used as an input for AVL Cameo. Given some target values, AVL Cameo calculates an optimized driveline calibration based on a calibration model. This calibration is evaluated by a simulation in the next iteration round. Throughout several iterations the design can be adapted and optimized already in a virtual environment. This process has also been described in detail in deliverable 603.011 Section 3.1.2 which describes the respective simulation tool AVL Cruise. A schematic overview of a possible calibration set-up for calibration is shown in Figure 5-2. Tools such as Simulink and Cruise would be adequate candidates for performing such simulations.

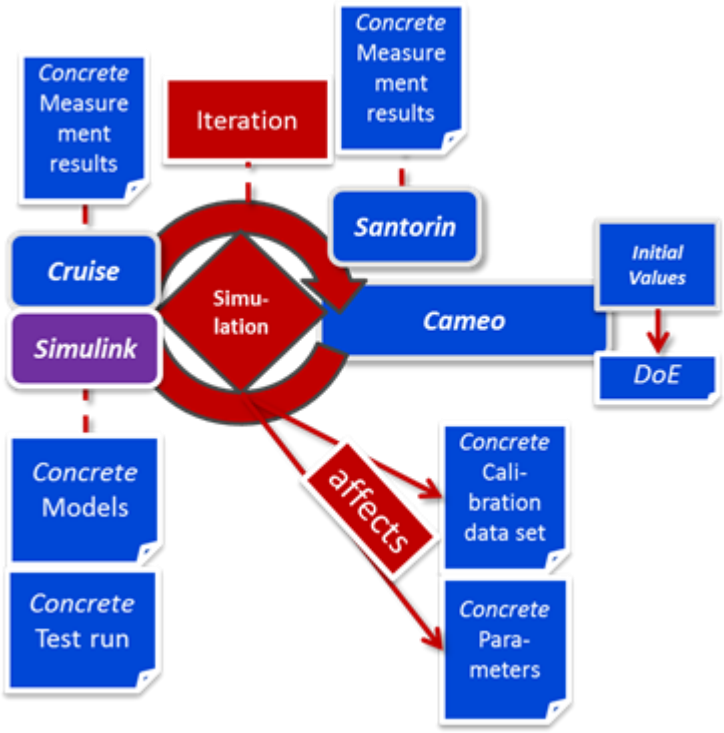

Figure 5-2: Tool-set up for a calibration iteration based on simulation

This iterative calibration process results in a high number of different calibration variants. Even more variants have to be handled if calibration data should be re-used over development cycles, whereas the simulation models are more and more replaced by physical components. Even more calibration variants come into play since each calibration cycle has to be started with an initial calibration data set, which is related to previous projects based on similar requirements. AVL CRETA is able to store and manage all these variants.

The related use case in WP3.4 has a strong focus on calibration tasks and calibration data re-use across vehicle development and test phases. Figure 5-3 illustrates for instance the migration from a calibration iteration based entirely on simulation to a test phase, whereas the simulated engine model is replaced by a physical instance of the related engine. This physical engine is then integrated in a testbed environment, with its own configuration tools.

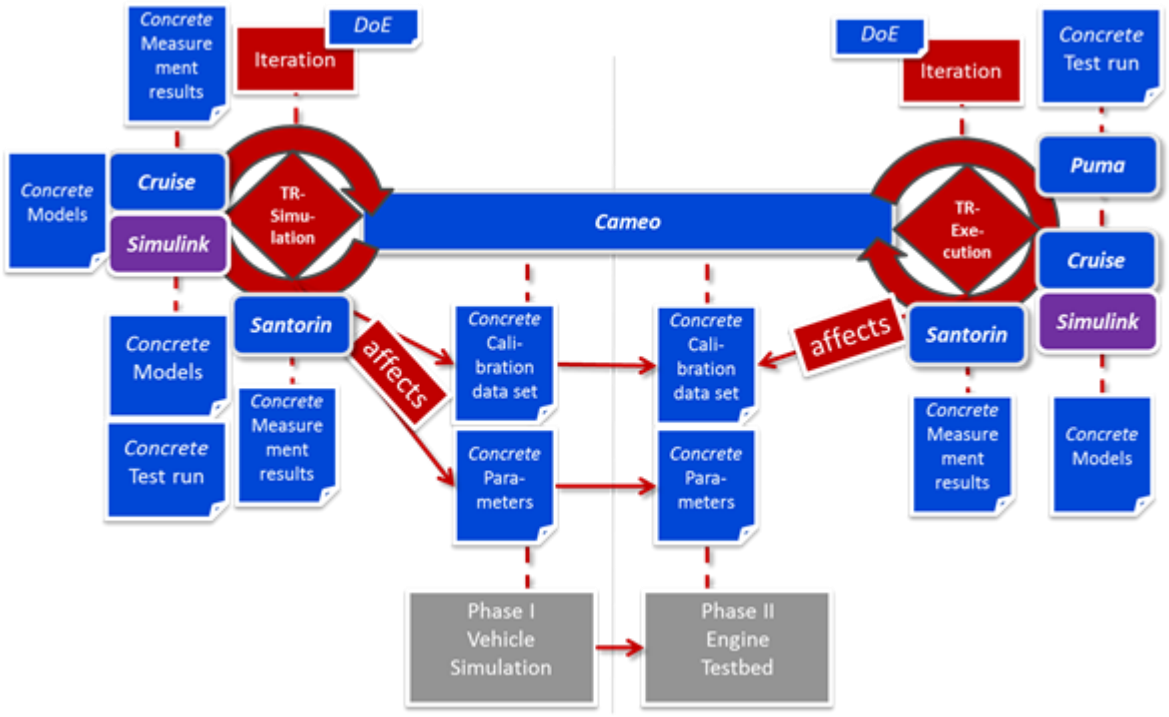

Figure 5-3: Calibration data migration from one testing phase to another

5.3 Integration within the IOS

According to the previous section, existing standards and related IOS challenges have to be evaluated and improved during the project. For instance, there is an ASAM standard for calibration data called ASAM MCD-2 MC.

A possible approach that deals with the linked-data concepts of OSLC and with existing ASAM standards is shown in Figure 5-4: A calibration tool (i.e. AVL Creta and/or Cameo in our case) is storing its data (e.g. calibration variants) in a proprietary format, but is also capable of exporting some aspects of this data to different standardized formats (such as ASAM MCD-2 MC for calibration data and ASAM ODS for measurement results). The exported data may be stored in a central data base such as the AVL data backbone concept (e.g. using AVL Santorin). OSLC adaptors on top of all involved tools and data providers, however, allow direct access to top level elements of the data artefacts applied in the engineering method about test and calibration iteration. These top level elements are elements of a high level OSLC resource model that complies with artefacts presented by this engineering method. A uniform workbench (such as the AVL Navigator tool) can then be used to navigate on a concrete instance of this OSLC model. Consequently, if the activities (of which the current engineering method is comprised) are embedded in this workbench properly, the usage of this tool would ensure that links are set properly and enables corresponding data navigation and reuse in related projects or later testing phases.

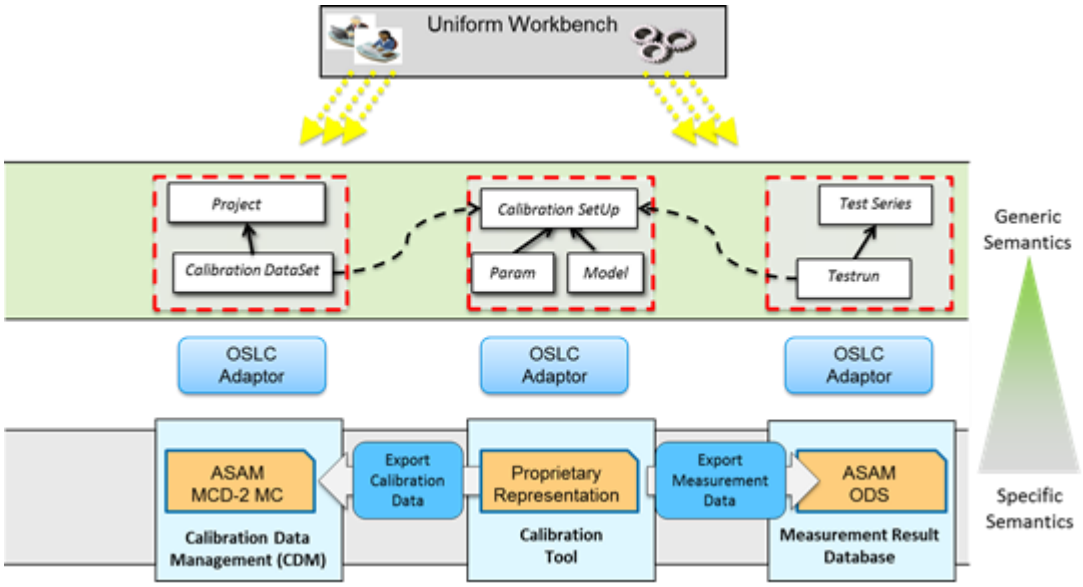

Figure 5-4: OSLC linked-data approach for calibration data management

Another main aspect for the optimization of calibration datasets is the continuous data exchange with simulation tools via such a data backbone approach. In order to provide consistent data throughout the development process, concepts for such data backbone will be developed as a separate brick in WP6.13. This task is closely related to this work package, because the interoperability of AVL CRETA and AVL Cameo with this data backbone has to be ensured. Figure 5-5 gives a rough over view about the general idea of such a data backbone in terms of the belonging AVL UC3.4.

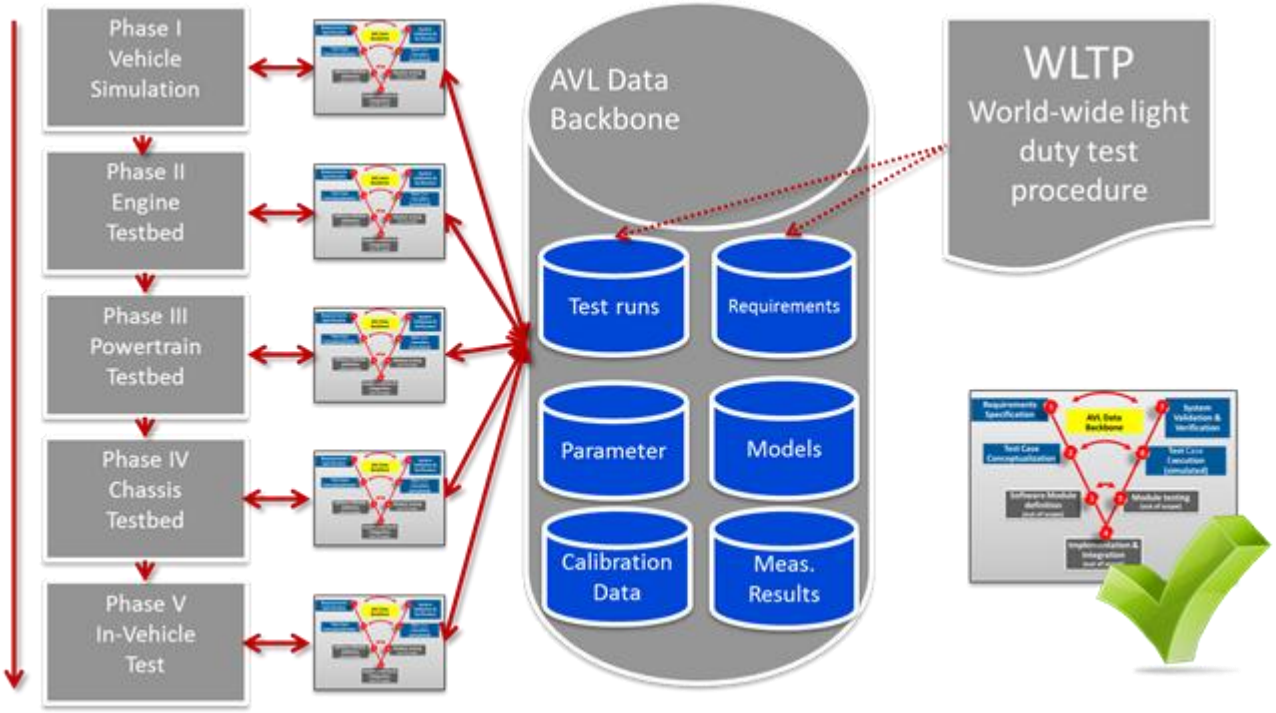

Figure 5-5: general idea of such a data backbone in terms of the belonging AVL UC3.4

The AVL Data Backbone acts as some kind of single-source-of-truth for all tools (including AVL Creta/Cameo) and data categories (including calibration variants) applied in all testing phases represented by different testing V-models (and thus different tools). With this concept, consistency among the development processes should be enabled and effective frontloading of development tasks become possible.

6 DSL Brick (Task 6.10.8)

This brick provides capabilities to design “Domain Specific Languages” and automatically generate code and other artifacts from such models.

6.1 Description

Name:	DSL Brick
Contact:	konstantin.keutner@siemens.com
Technical Information The brick is based on the mature, well-known Xtext technology. This technology chain is based on EMF (Eclipse Modelling framework), which is part of the Eclipse ecosystem. Alternatives which extend the state of the art are being considered as well, including Gtext which is based on GLL (Generalized LL parsing). This gives more concise syntax definitions, closer to semantics.	
Operation System/platform	Portable, based on the Java technology.
Version	Xtext/Xtend 2.4.3
Type of Input Data	Domain specific models. These represent specific information for the particular domain, and can be freely defined. Integrations to other models can be achieved with model-to-model transformations.
Type of Output Data	Very flexible output formats are possible: Text-based structured output (such as source code) XML Custom connectors to other data sinks.
Dependencies	Eclipse 3.5 (Kepler)
License	Eclipse Public License
Additional information	

Domain Specific Languages help to raise the level of abstraction, thus getting closer to requirements than to code. In the area of software development, they can improve the efficiency of code development for the variability hotspots of a domain. The development and use of DSLs and generators for the implementation of safety-critical systems poses particular challenges with regard to selection and usage of tooling in order to develop safety-certifiable high-integrity code.

This brick will provide a domain-specific language tool suite comprising existing open source tooling, additionally developed components, and guidelines specifically tailored towards safety-critical embedded systems development. This will facilitate the efficient development of domain specific languages and code generators in the different use cases, with a particular focus on safety-critical code generation and the safety certification process.

6.1.1 Tool/method description: what will you find in the DSL brick

Xtext is an open-source framework for developing programming languages and domain-specific languages (DSLs). Unlike standard parser generators, Xtext not only generates a parser, but also a class model for the Abstract Syntax Tree and a fully featured, customizable Eclipse-based IDE.
To specify a language, a user has to write a grammar in Xtext's grammar language. This grammar describes how an Ecore model is derived from a textual notation. From that definition, a code generator derives an ANTLR parser and the classes for the object model. Both can be used independent of Eclipse.

Into the bargain, Eclipse-based IDE integration is generated. That IDE offers e.g.: Syntax coloring, Code completion, a restricted form of static analysis, Outline view, Source-code navigation, Indexing, Compare view, Hovers, Code folding and Rename refactoring.

Xtext languages and the IDE are highly configurable, as the language infrastructure and the IDE are wired up using dependency injection and Guice. The default components can be easily replaced by binding customized ones instead.

Since version 2.0, Xtext facilitates the development of domain-specific languages for the Java Virtual Machine, referring to and compiling to Java artifacts with tight integration into Eclipse's Java Development Toolkit. A reusable expression languages library enables rich modelling right within the DSL.

A code generator written in Xtend can be easily hooked in for any language. For JVM languages, it is enough to map the DSL concepts to Java artifacts to get holistic Java integration. An alternative interpreter is also available.

6.1.1.1 Installing and using the DSL brick

Xtext can be downloaded and installed using either

- 'Full Eclipse' installation – a pre-configured Eclipse distribution can be downloaded which has already all the necessary plug-ins installed.
- 'Update Sites' installation – if you have an Eclipse installation running, you can use the Eclipse update site mechanism to additionally install the Xtext plug-ins.

Both mechanisms are described in detail on the Xtext website http://www.eclipse.org/Xtext/download.html. Follow the instructions in the relevant section.

6.2 Improvements in CRYSTAL

In collaboration with Philips HealthCare, a DSL using the Eclipse Modeling Framework (EMF) has been defined, and in particular Xtext. Based on the good experiences, in the CRYSTAL project we are planning to develop more DSLs using these technologies. After discussions with various developers and managers from Philips HealthCare, we have identified a couple of open issues:

1. Modularity of DSLs, based on reusable fragments:

- Many DSLs share some fragments that deal with common concepts, it is desirable to be able to reuse such fragments (meta-model, grammar, validation, etc.). For example, a DSL fragment that can easily be used for all kinds of expression languages, which differ mainly in terms of the non-terminals. Xbase seem to be a proposal, but it looks too big to be used as input for code generation.

2. Modularity of DSL instances, based on import mechanisms:
    - DSL instances describing industrial systems can become quite big, it is desirable to decompose them using a kind of import mechanism. There are many styles of import mechanisms (e.g., Java, C++) also depending on whether they work recursively and whether they can deal with cycles (Xtext has two predefined mechanisms). Adding such mechanisms to a DSL impacts grammar, scoping, validation, code generation, etc. The limitations of such mechanisms will be studied in order to have reusable techniques to build such mechanisms properly.
3. Integration of textual and graphical editing:
    - In the context of EMF, GMF focuses only on graphical editing, and Xtext focuses only on textual editing. It is desirable to edit parts of a DSL instance in a graphical way and other parts in a textual way; some parts may even be edited in multiple ways.
4. Scalability:
    - DSL instances describing industrial systems can become quite big, It is desirable that the DSL language infrastructure scales properly. This includes the performance of parsing, scoping, outline tree, validation, etc. For example, there may be implementation guidelines, caching approaches, or information about the (in)efficiency of generic default implementations.
5. Debugging DSL instances:
    - Generating code from a DSL is one thing, but what to do if the generated code doesn't work properly (in the context of some other code)? It is desirable to debug the generated code at the abstraction level of the DSL. For example, there exists an integration of Visual Studio with MetaEdit+, but not with EMF.
6. Definition of semantics
    - The current generation of language workbenches, Xtext and Eclipse is one of them, are very syntax oriented. There are very restricted facilities to define static semantics; in fact only identification can be defined with Xtext in combination of simple scoping rules. If more static semantic rules have to be defined, one has to do this in Java. The definition of the dynamic semantics of a DSL is entirely lacking. The dynamic semantics is mostly expressed in the translation of the DSL into some general-purpose language.

Additional issues look at:

1. Integrated tooling, additional components and application guidelines for DSL and generator development in safety-critical systems.
2. Analyze contributions/problems of DSL approach with certification:

- Failure modes of generative approach?
- Generation of certification artifacts possible?
- Comparison with other approaches (embedded)

6.2.1 Case Study coverage

Use case WP4.1 (Philips HealthCare) has identified Domain Specific Languages (DSLs) as a way to define, represent, validate and transform models of system Modelling. It will be applied in the iterative development of systems where patient safety is absolutely critical.

This use case is not about the development of a single DSL; it is likely that multiple DSLs will be developed, emphasizing different aspects of the system. The following requirements are relevant in this application:

1. Ability to quickly develop and modify a DSL (e.g., guidelines, semantic building blocks);
2. Ability to debug DSL instances (in particular in the presence of code generators);
3. Ability to visualize parts of textual DSL instances;
4. Scalability of the Xtext technology to industrially-sized DSL instances.

In addition, an indirect contribution to WP4.1 from the DSL Brick is made via task 6.3.6 on improving the POOSL tools. The improved POOSL tools will be based on Xtext technology, and hence POOSL should be considered as a (rather complex) Domain Specific Language (DSL). The following requirements are relevant in this application:

1. Ability to split large POOSL models into smaller files (such as libraries);
2. Ability to edit certain parts of POOSL models in a textual way and others in a graphical way;
3. Scalability of the Xtext technology to industrially-sized POOSL models.

6.3 Integration within the IOS

Need to develop a vision of possible interfaces (inputs and outputs).

Integration activities:

- Integrate existing DSL and generator tooling (Eclipse, EMF, Xtext, Xtend, and others) and provide it to the use cases. Other model transformation languages for EMF such as Epsilon could be candidates as well.
- Alignment with Task 6.10.2 System Family Engineering to tailor existing tooling and its usage to safety and certification process demands and avoiding pitfalls.

Relate (elements in) input models and output artifacts to a requirements tool (or a higher-level design model).

7 AUGE Brick (Task 6.10.9)

7.1 Description

Name:	AUGE
Contact:	carlos.zubieta@orbital-aerospace.com
Technical Information: This brick will be a standalone Software application with graphical HMI allowing the user to browse a set of Requirements stored in an IOS-compatible Requirements Management Server (i.e. DOORS) and generate automatic tests in generic IOS test language.	
Operation System/platform	Linux
Version	1.0
Type of Input Data	IOS Formal Requirements, IOS Requirement Changes
Type of Output Data	IOS Test syntax
Dependencies	N/A
License	Terms and conditions agreed in CRYSTAL APCA
Additional information	N/A

7.1.1 Tool/method description: what will you find in AUGE

Currently, software engineering processes in sectors such as Space and Avionics are facing important variability challenges when integrating Requirements Management Systems (i.e. DOORS) data modules to Verification & Validation artifacts to perform required testing and traceability.

Typically, a high-level requirements module must be traced to a related low-level module, and from this one, an additional module refined with convenient test data and syntax is produced and exported to a third-party format (i.e. Microsoft Excel). This set of data is somehow parsed and imported into an Automatic Testing Tool (i.e. TestStand) in order to proceed with V&V. Thus, the whole process involves heavy data processing which provides no added-value and increases significantly costs and time.

AUGE is SW standalone application that will reduce such costs in Verification & Validation campaigns by achieving automatically test generation from requirements. The tool shall be an independent entity totally driven by IOS-compatible data interfaces and formats, enabling the integration of any kind of Requirements Management Systems and Automatic Testing Tools (provided that these bricks are IOS-enhanced, via plugins, adapters or internal modifications).

AUGE shall focus in Space software development use case, providing specific integration with ESA ECSS-E-40 standard. This way, generated test shall be aligned with space certification criteria and could be provided as relevant evidences in ESA software certification process with minimum effort.

7.1.1.1 Installing / Using method AUGE

AUGE shall be a GNU/Linux native application, distributed as a standard Linux package such as .rpm or .deb file. The package will include all software dependencies (if any) to ease installation process to final user.

In order to use AUGE, a predefined configuration file shall be set by user with all relevant IOS settings such as:

- Requirements Management Server(s) URI(s).
- User credentials (password, profile).

The user shall be able to select a predefined server and browse permitted Requirements modules according to his profile. Once a requirement is selected, the user shall either:

- Preview the Requirement. A new dialog window shall display some requirement data, as defined in OSLC “Linking Data via HTML User Interface” specification.
- Generate Test. The Requirement data shell be processed by the application. If the data syntax is correct the requirement will be considered mature and an output text file shall be generated containing an automatic test. Otherwise, the requirement shall be considered not mature and an error message shall be displayed.

7.1.1.2 TEST AUGE installation / Example of usage of the Method

The installation is straightforward provided that a standard Linux package will be distributed.

Example:

For Debian Linux environments, a .deb file shall be packaged. To install, execute the following command:

- sudo dpkg –i auge.deb

To run the application, execute the following command:

- auge

7.2 Improvements in CRYSTAL

7.2.1 Case Study coverage

AUGE is aligned with Use Case 2.5 in Space domain, leaded by Thales-Alenia Space España. This use case focus on integration of Space hardware with embedded software provided by external manufacturer.

AUGE tool will enable system integrator to get seamless ISVV (Independent Software Verification and Validation) by leveraging variability challenges aroused from incompatibilities in partners applications and data formats.

TASE inputs shall be required in the future to obtain automatic tests aligned with ESA-ECSS-E40 standard.

7.3 Integration within the IOS

AUGE shall be designed to totally integrate with IOS. Moreover, all inputs and outputs shall be IOS-related so no linkage to specific commercial tools will be required. The following IOS workgroups have been identified to cover AUGE integration requirements:

- Formal Requirements Management
- Change Management
- Documentation Generation

8 Terms, Abbreviations and Definitions

8.1 CRYSTAL-specific managerial abbreviations:

CO	Confidential, only for members of the consortium (including the JU).
CRYSTAL	**CR**itical S**YST**em Engineering **A**cce**L**eration
D	Demonstrator
O	Other
P	Prototype
PP	Restricted to other program participants (including the JU).
PU	Public
R	Report
RE	Restricted to a group specified by the consortium (including the JU).
SP	Subproject
WP	Work Package

Table 8-1: CRYSTAL-specific managerial abbreviations

8.2 ACRONYMS, used in this deliverable:

ASAM	Association for Standardisation of Automation and Measuring Systems
CNG	Compressed Natural Gas
DSL	Domain Specific Language
EMF	Eclipse Modelling framework
ESA	European Space Agency
GMF	Graphical Modeling Framework
GLL	Generalized LL parsing
HMI	Human Machine Interface
HTML	Hypertext Mark-up Language
IDE	Integrated Development Environment
IOS	Interoperability Specification
ISVV	Independent Software Verification and Validation
JVM	Java Virtual Machine
LPG	Liquid Propane Gas
MC	Measurement & calibration
MCD	Measurement, Calibration, Diagnostics
ODS	Open Data Services
PDF	Portable Document Format

POOSL	Parallel Object-Oriented Specification Language
TFMS	TestFactory Management SuiteTM
URI	Uniform Resource Identifier
V&V	Verification and Validation
WLTP	World-wide harmonized Light-duty Test Procedure

Table 8-2: Terms, abbreviations and definitions

9 References

Tool	URL	Related Bricks
Eclipse	www.eclipse.org	T6.10.3, T6.10.8
Guice	https://code.google.com/p/google-guice/	T6.10.8
Xtext	www.eclipse.org/xtext	T6.10.8
EMF	https://www.eclipse.org/modeling/emf/	T6.10.8
Xtend	https://www.eclipse.org/xtend/	T6.10.8
GMF	http://www.eclipse.org/modeling/gmp/	T6.10.8

Table 9-1: Tools related to the Bricks