Doc ID	18112301R05
Doc Creation Date	27 MAR 2018
Doc Revision	05
Doc Revision Date	12 DEC 2018
Doc Status	Released

Work Package	Deliverable ID
WP6: implementation and integration of I-MECH platform	D6.1: Test benchmarking and strategy

Executive summary

This document covers the strategy to be applied for the evaluation of the different component test and the integration test. The benchmark for performance testing will be specified in this report. The report also summarized benchmarking of subcomponents reused from other linked project.

(Main)Authors	Carlos R. de Yurre (AOTEK). Michael Walsh (Tyndall)

Keywords

Test, verification, validation, integration, platform, building block.

Coordinator	Fagor Aotek
Tel.	0034 943 039 805
E-Mail	yurre@aotek.es
Internet	www.aotek.es

(Open) Issues & Actions

Open Issues (and related actions) that need central attention are part of a file called "IAL - Issues & Action List – WP6" which can be found in the Google Drive Partner Zone.

ID	Description	Due Date	Owner	IAL ID
OI-01	There is little or no description of time scheduling.	12/2018	Carlos Yurre	
OI-02	The definitions lack state machine for life cycle: when to initialize integrators, when to initialize Fieldbus, initialization order for BBs….When defined, the verification must assure that important functions do exist for them.	12/2018	Carlos Yurre	
OI-03	It is not clear how BB10-BB11 relate with the rest of BBs. It seems that they both build a tandem where the rest of BBs can reside, but we should try to standardize somehow the services. Verification must assure that BB10-BB11 constraints are compatible with I-MECH control BBs requirements	12/2018	Carlos Yurre	

Document Revision History

Revision	Status	Date	Deliverable lead	Description of changes	IAL ID / Review ID
R01	Draft	27-MARCH-18	FAGOR AOTEK	Initiate	
R02	Draft	20-AUGUST-18	FAGOR AOTEK	Include reqs. From WP2, WP4	
R03	Draft	19-SEPTEMBER-18	FAGOR AOTEK	Rewrite after revision of scope	

R04	Draft	17-OCTOBER-18	FAGOR AOTEK	Include proposals, FMI, XIL	
R05	Release	12-DECEMBER-18	FAGOR AOTEK	Corrections of internal review	

Contributors

Revision	Acronym of Partner	Contributor	Description of work
R01	EDI	Kaspars Ozols, Rolands Shavelis	Contribution to sections 2, 4, 5 and 6. Overall feedback on document.
	SISW	Pacome Magnin	Improvement of section 5.4 to 5.8, V&V tools and methods (adding FMI related details, PiL techniques, HiL techniques)
	SCC	Hans Kuppens	Appendix B
R02	FAG	Carlos Yurre	Requirements from WP2 and WP4
R03	FAG	Carlos Yurre	Feedback on draft
R04	FAG	Carlos Yurre	Feedback on draft
R05	TNI	Michael Walsh	Contribution to section 8
	TNL	Marc van Eert	Contribution to section 5.11
	FAG	Javier Arenas	Feedback on draft

Document control

	Status	Release							
	Revision	05							

Reviewer Name	Role	Selection							
Hans Kuppens	Internal peer reviewer (Sioux CCM)	X							
Marc van Eert	Internal peer reviewer (Technolution)	X							

File Locations (cross reference to I-MECH documents)

Via URL with a name that is equal to the document ID, you shall introduce a link to the location (either in Partner Zone or CIRCABC)

URL	Filename	Date
2017102001R01	I-MECH Requirements Table	21-OCT-2017

Abbreviations & Definitions

Abbreviation	Description
BB	Building Block
DoW	Description of Work
HAL	Hardware abstraction layer
HIL	Hardware in the Loop
HW	Hardware
MIL	Model in the Loop
PIL	Processor in the loop
XIL	Any one of MIL,SIL,PIL,HIL
RTOS	Real Time Operating System

SIL	Software in the Loop
SW	Software
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit

Table of contents

1. Schedule

1.1 Deliverable Completion Schedule

The timing diagram provided below details a suggested completion schedule for the associated deliverables in WP6. A two phased based schedule is envisaged with key validation and verification deliverables submitted toward the end of year 2 and final reports to be compiled nearing project completion.

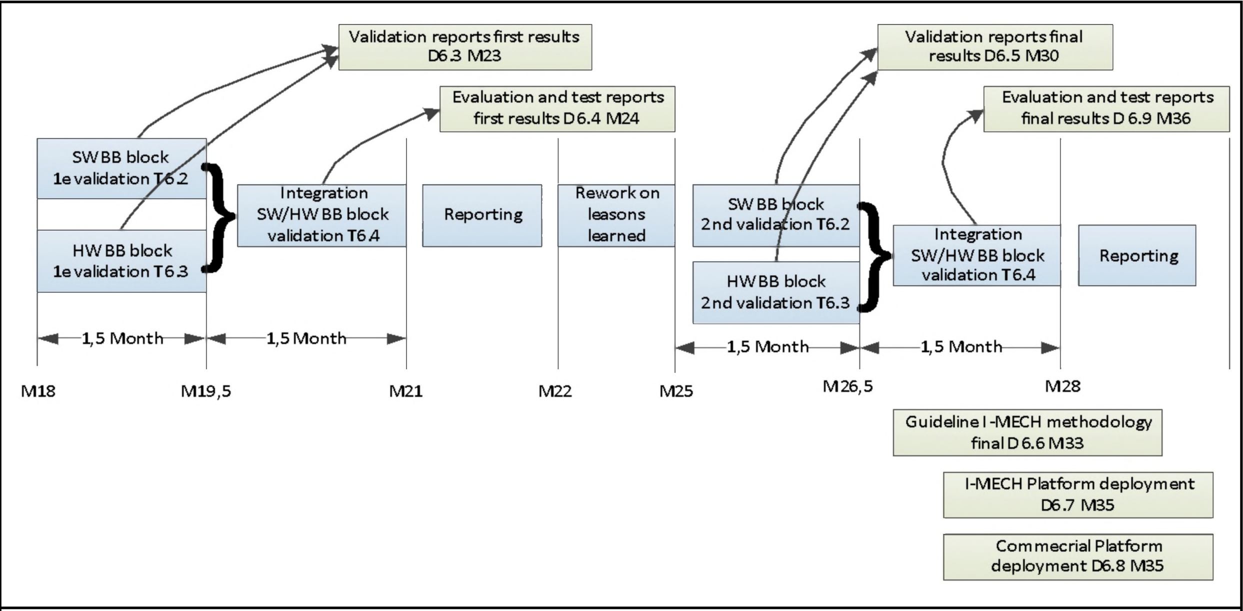

Validation reports first results D6.3 M23
Evaluation and test reports first results D6.4 M24
Validation reports final results D6.5 M30
Evaluation and test reports final results D 6.9 M36
SW BB block 1e validation T6.2
HW BB block 1e validation T6.3
Integration SW/HW BB block validation T6.4
Reporting
Rework on leasons learned
SW BB block 2nd validation T6.2
HW BB block 2nd validation T6.3
Integration SW/HW BB block validation T6.4
Reporting
1,5 Month
1,5 Month
1,5 Month
1,5 Month
M18
M19,5
M21
M22
M25
M26,5
M28
Guideline I-MECH methodology final D 6.6 M33
I-MECH Platform deployment D6.7 M35
Commecrial Platform deployment D6.8 M35

Figure 1

2. Synopsys

From Description of Work, DoW, “the **Task 6.2 Validation of I-MECH SW components/platform**: Library of SW functional blocks implementing **the developed algorithms from WP4 and WP5 will be validated against requirements** developed in WP2 and **precised** in Tasks 4.1 and 5.1.
Following agile W approach, the iterative validation will be reported in 3 stages (**D6.1**, D6.3 and D6.5), each followed by integration process (Task 6.4). The focus will be put on functional and performance

requirements of all stakeholders (SMEs working in mechatronic innovations (ROV, RED, TECO), control system integrators (ITML), manufacturing end users (GMV, PHI, COR, VIS)) and whether the results are efficient, flexible and general enough in order to fulfil their needs.
The ability to deploy SW intelligence (UWB, EDI, TEK, BUT, UNIBS, UMO, TNI) onto commercial platforms will be tested (see consequential Task 6.5). The Multi-OS software stack (i.e, hypervisor and RTOSs) for the platform will be tested as well (EVI). The SW validation results will be continuously documented and delivered to the I-MECH development process."

From DoW, the deliverable D6.1 covers the strategy to be applied for the evaluation of the different component test and the integration test. The benchmark for performance testing will be specified in this report.
This document is based on I-MECH available requirements at the moment (D3.1, D4.1, D5.1).
The work on the Tasks 6.2 and 6.3 should take into account those requirements and influence them as to fulfil validation.
A common, expressed and exposed view of the whole architecture of an I-MECH platform is mandatory. The internal structure of Building Blocks up to the level needed for validation must also be expressed in some way.
It is relevant for the sake of clarity that the terms used for logical entities are defined and consistently used through any validation document. It is not clear that every functionality can be tested.
Work on Task 6.1 leads Building Block definition.

As for D4.1, this deliverable is strictly related to the activity of WP4 "Control Layer design and development", which aims at developing centralized and decentralized motion control strategies for mechatronic systems.
Then, D4.1 (D4.2) can be used to test the results of the building blocks directly related to the control layer, that is:
BB6 "Self-commissioning velocity and position control loops",
BB7 "Vibration control module",
BB8 "Robust model-based multivariable control"
BB9 "Iterative and repetitive control module"
and of those building blocks (BB10 "Control Specific Multi-many core Platform" and BB11 "RTOS for multi-many core platform") which are related to the HW/SW platforms which allows the implementation of the advanced control algorithms.

Also from D4.2, requirements are divided in Performance, Technical and Realization.
Technical requirements are directly connected to Functionality and could also be called Functional requirements. These answer the "What's the BB functionality?" question, and a list of BBs will be included with a description for how everyone complies with the V&V requirements.

I-MECH hardware building block components will be validated against the requirements developed in WP2 and further refined in Tasks 3.1 and 3.2.

BB-1 Platform for Smart Sensors with Advanced Data Processing
BB-2 Real-time wireless sensors providing complementary feedback information
BB-3 Robust condition monitoring and prediction diagnostics
BB-4 High Speed Vision
BB-5 High performance servo amplifier design
BB-10 Development / selection of control specific multi-many core platform

System behaviour layer integration and connectivity requirements and specification for each of the components are sources in Tasks 5.1 and 5.2.

3. Definitions for Verification and Validation

3.1. Verification and Validation

Verification and validation are independent procedures that are used together for checking that a product, service, or system meets **requirements** and specifications and that it fulfills its intended purpose.

Verification is the process of evaluating a system or component, to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Its aim is building the system right.
It answers the questions: “Does the system meet its requirements?”, “Am I building the product right?", "Did I build what I said I would?"

Validation is the process of evaluating a system or component during or at the end of the development process, to determine whether it satisfies specified requirements. Requirements Validation is the process of confirming the completeness and correctness of the requirements. Its aim is building the right system. Validation determines that a system does all the things it should and does not do what it should not do.

It answers the questions: “Are the system design requirements correctly defined and mean what we intended”, “Am I building the right product?”, "Did I build what I need?"

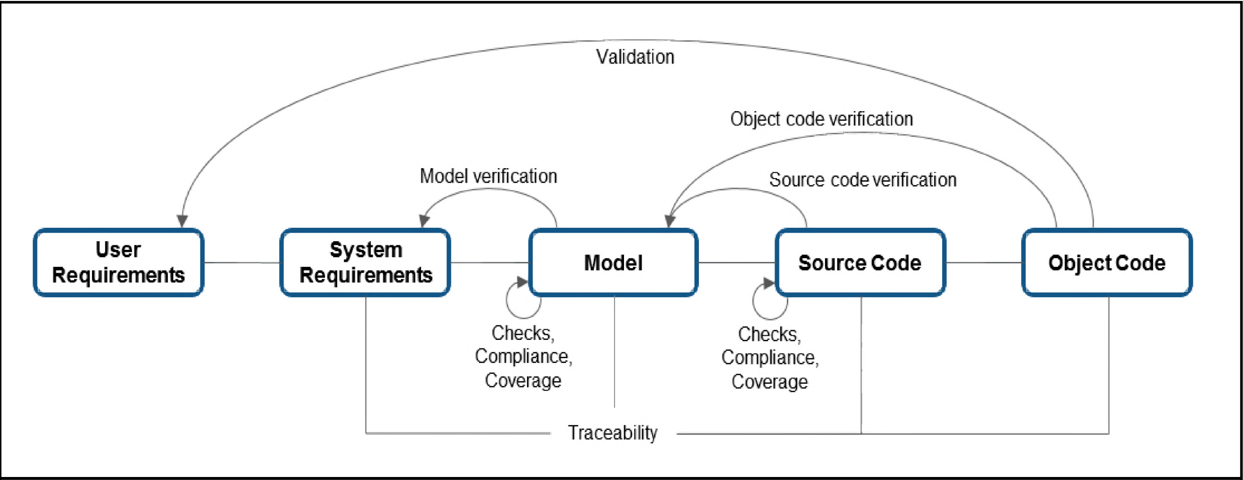


Figure 2

Validation is done against user requirements. This means that for validation of I-MECH BBs, the platform should have functionalities related to layer 3, that is, a PIL, HIL or deployment phases. The communication with the user (eg. OPC UA) must be present and interaction defined must be validated.

4. Scope of deliverable

The document addresses the methodology for validation and verification of Building Blocks.
On a first approach, these BBs have common characteristics derived from being an I-MECH Building Block that must be verified, and specific behaviours from the requirements of every one, derived mainly from D4.2 for SW Building Blocks and from D3.2 and D5.2 for HW Building Blocks.

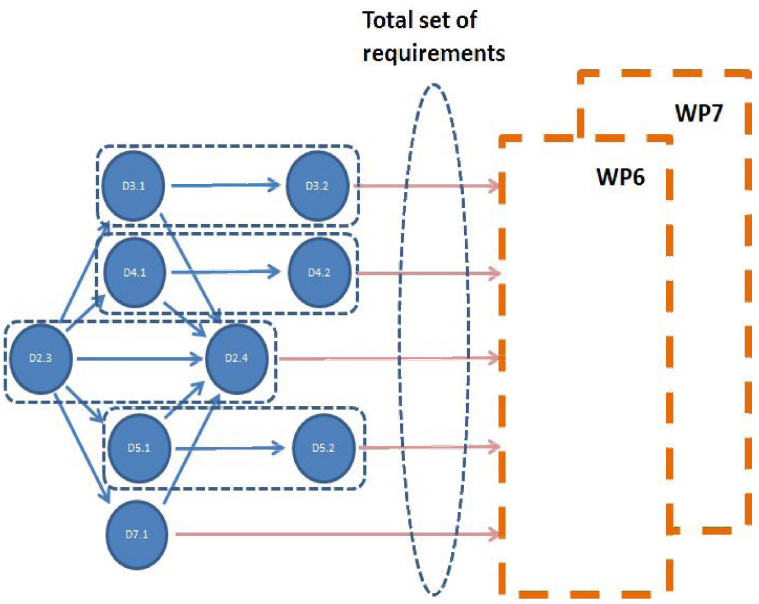


Figure 3

The figure 3 is borrowed from the WP6 kick-off meeting presentation. It shows the dependence graph between different requirements deliverables. The same set of requirements are the input for pilot testing. WP6 refers to testing the Building Blocks, where Task 6.2 focuses on software BBs whilst Task 6.3 addresses hardware BBs. This figure also shows that there is a difference between verification and validation and test, while sharing requirement documents.

That's an important division and should be kept in mind when dealing with requirements that can't be tested without the whole platform and hardware. This document refers frequently the documents named in the figure (D2.3, D2.4, D4.2….). It is supposed that they have been read before.

The following figure (figure 4) illustrates the requirements generated in each of the work packages for each of the associated building blocks. It also highlights where verification and validation will take place for each of these requirements.

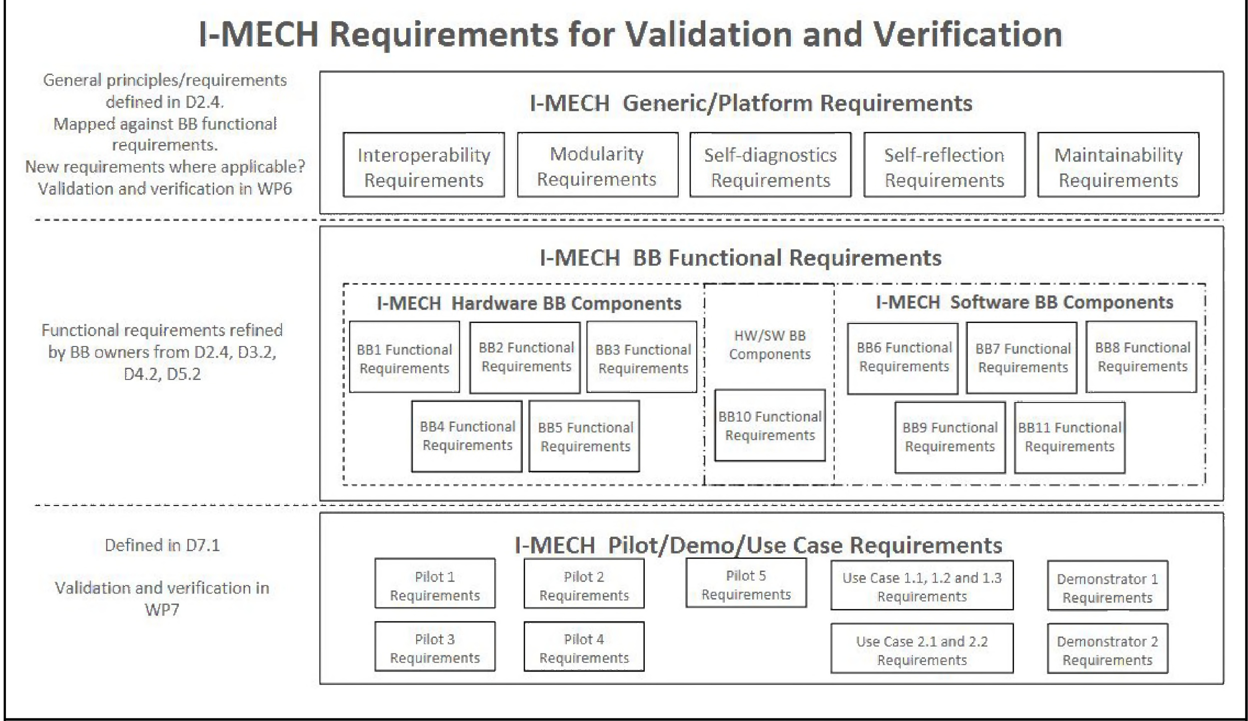

I-MECH Requirements for Validation and Verification
General principles/requirements defined in D2.4. Mapped against BB functional requirements. New requirements where applicable? Validation and verification in WP6
I-MECH Generic/Platform Requirements
Interoperability Requirements
Modularity Requirements
Self-diagnostics Requirements
Self-reflection Requirements
Maintainability Requirements
Functional requirements refined by BB owners from D2.4, D3.2, D4.2, D5.2
I-MECH BB Functional Requirements
I-MECH Hardware BB Components
BB1 Functional Requirements
BB2 Functional Requirements
BB3 Functional Requirements
BB4 Functional Requirements
BB5 Functional Requirements
HW/SW BB Components
BB10 Functional Requirements
I-MECH Software BB Components
BB6 Functional Requirements
BB7 Functional Requirements
BB8 Functional Requirements
BB9 Functional Requirements
BB11 Functional Requirements
Defined in D7.1
Validation and verification in WP7
I-MECH Pilot/Demo/Use Case Requirements
Pilot 1 Requirements
Pilot 2 Requirements
Pilot 5 Requirements
Use Case 1.1, 1.2 and 1.3 Requirements
Demonstrator 1 Requirements
Pilot 3 Requirements
Pilot 4 Requirements
Use Case 2.1 and 2.2 Requirements
Demonstrator 2 Requirements

Figure 4

5. I-MECH methodology for Verification and Validation of Building Blocks

Building Blocks and layers are described in the DoW. In what follows we will assume all BBs will have some common functionality. These common functionalities, related with the I-MECH objectives, build up an I-MECH Building Block. A wider description of abstract building blocks can be found in D2.3, Ch.2.2.1.1

I-MECH Building Block definition: ***a logical block with some exposed behaviour or functionality that fulfils a minimum set of the defined I-MECH requirements. In what follows it is called BB.***
I-MECH refers mainly to model based system engineering. This process is described in D2.3, in chapter 2.2.1.3. In the V model, V&V standing for verification and validation, **the methodology**

implies that requirements are written for the system and hierarchically for the subsystems.
Simulink is a platform of very broad acceptance and most of BBs will exhibit a Simulink interface. However, simultaneously, **other interfaces based on open standards will be introduced in order to assure interoperability between BBs**.

5.1. Rationale under V&V of Building Blocks

The aim for a BB developer is to offer a module, possibly including hardware, which can be easily integrated in a system that follows the Model Based design paradigm. Passing the Verification and Validation phases assures that such a system has been modelled and that its model can be used seamlessly in combination with other modules in simulation. This warranties then interoperability at simulation level.
But this is just part of the work. To run on the target the code can be generated from the simulation platform (the preferred method) or be included as DLL, VHDL...or any other binary form. Verification can at least assure that there is code available for the target processor, as declared by the BB developer.
And finally, Validation for the system designer includes assuring that the chosen BB is the right one for the higher level requirements. This is tied to the functional requirements and benchmarking. For instance, that a sensor has the needed bandwidth and resolution, and that the information comes with a delay compatible with its intended use. Verification of functional requirements assures that at least that information is present. What cannot be assured is that the figures are enough for the system. This is a high level validation phase.
At least in PIL and HIL phases, connection from external tools is needed. Accessing variables, parameters and tunable parameters is expected to be done. This can be viewed as a validation phase, as long as the names of the variables and namespaces (semantics) are declared. Verification will tell whether the namespace does exist and validation that declared data are available.
All these V&V activities should warranty that using a module, named as an I-MECH BB, integrates seamlessly in Model Based Design, from simulation to deployment. And this is the base of the I-MECH methodology.

5.2. I-MECH methodology

The I-MECH methodology involves all phases for engineering and relies on virtual prototypes for system and control. Model-in-the-loop (MIL), software-in-the-loop (SIL), processor-in-the-loop (PIL) and hardware-in-the-loop (HIL) must be considered. Rapid Prototyping can be an alternative or complement to HIL in many cases when dealing with control blocks.
I-MECH methodology must warrant that any BB that passes the verification and validation phases related to architecture meets the objectives of interoperability, modularity, …, and then

- can be connected to other I-MECH BBs
- can be used as a model in a model based design framework and toolchain
- can be deployed on at least a target that conforms to I-MECH standards (being an I-MECH BB11)
- can be accessed at run time through layer 3 interface
- can be configured through layer 3 interface

But it is important to remember some facts:
- For Model Based Design, both MIL and PIL are mandatory at some point during V&V and integration.
- I-MECH's aim is oriented to deployment where more features need to be added not addressed by the I-MECH BBs. Notably, coordinating calls and providing services to virtualize field buses.

Verification will test:
- that the Functional Mockup exists.
- that it is compliant with FMI standard (with a compliance checker)
- that description of block aims and name is included in the FMU description xml.
- that a preferred namespace is defined for OPC UA communication.

As it has been defined before, an I-MECH BB implements parts of the I-MECH functionality. Given the different disciplines involved, three types of modules can be differentiated:
- mainly software functionality (i.e. control algorithms)
- mainly hardware functionality (i.e. sensors and actuators)
- a hardware processing platform to deploy the software functions and connect to sensors and actuators.

All three parts are needed to build a control system. The goal of methodology for verification and validation is the same: to validate that the promised functionality is provided by the BB. However, the way to proof this differs for hardware and software parts.
For software blocks a MIL tests to prove the functional is working is mandatory.
For hardware blocks a HIL tests to prove hardware, software compatibility is mandatory.
For the processing platform validation is mandatory that software building blocks can be deployed and that sensors and actuators can be used.

5.3. Higher level requirements, objectives

Some common requirements for architecture of Building Blocks derive directly from the I-MECH **objectives** in DoW :

BBs are interoperable, what means having a very clear and exposed behaviour regarding inputs, outputs and internal states and transitions. The validation task must read this information and test any state and transition. In fact, for every state, both available and don't care transitions shall be known and tested.

A standard way to share or exchange models, including the possibility of cosimulation is Functional Mockup Interface https://fmi-standard.org/

A Functional Mockup Unit (FMU) is a .zip file containing several directories and source or binary code and addressing one of two targets: Model exchange or Cosimulation. The specifications are available free at https://fmi-standard.org/downloads/ .

There is also a free tool for compliance checking:
https://trac.fmi-standard.org/browser/branches/public/Test_FMUs/Compliance-Checker

FMI defines two different interfaces, one for model exchange and another for co-simulation. At least one of them must be present. The compliance checker verifies that an FMU has the expected files and functions defined in the standard. From this point of view, defining FMI as the standard interface definition for I-MECH has the advantage that interoperability of models of the BBs is granted for all the simulation tools that support importing FMUs and verification of the formal aspects can be done with the compliance checker.

I-MECH BBs will use specification from FMI for all simulation cases. If a BB will not be a FMU (with good rationale), a equivalent Simulink interface specification should be defined.

BBs are modular. This means that aggregations of BBs are also BBs or at least have a BB behaviour. In fact, this means, and is a requirement, that any composition of BBs that wants to export an I-MECH BB behaviour must comply with all the requirements for V&V.

To this end, there is an ongoing project under the Modelica Association umbrella:
https://www.modelica.org/projects

The System Structure and Parameterization of Components for Virtual System Design project (SSP) addresses the problem of composition of FMUs.

The document *10_2017_FMI-Usermeeting_SSP-StatusandPlans.pdf* defines objectives:

- Define a standardized format for the connection structure of a network of components (FMUs in particular).
- Define a standardized way to store and apply parameters to these components.

The developed standard / APIs should be usable in all stages of development process (architecture definition, integration, simulation, test in MiL, SiL, HiL).

While work seems not being concluded, the proposal is following this project and possibly adhering to the standard file formats defined there. But from the point of view of V&V of I-MECH BBs, the requirement keeps the same, the composite block must behave like a BB and have the FMI defined files.

BBs have (self-)diagnosis. If this is a mandatory requirement, the validation phase must have some means to verify that diagnosis works. This can be done by injecting an error somehow or putting the system to an error state. This is closely related with **validation**.
Having test points and means to record data is a common feature for diagnosis and validation. From the point of view of a BB or an FMU when dealing with simulation, accessing the model and its variables both for injection of signals and data recording is mandatory. Self diagnosis can be considered as diagnosis done with software provided by the BB builder (an example is provided in appendix B). Comparing output after injecting signals to prescribed patterns needs a set of services. This feature has been addressed by the Association for Standardization of automation and Measuring Systems (ASAM) https://www.asam.net/ in the project ASAM XIL Maintenance, where XIL denotes the full process (X = M,S,P,H) . The project has adhered to FMI for model exchange and is referenced in the FMI web page:
https://fmi-standard.org/related/ .
Specifications are found there as a link. This seems to be an ongoing project also, and the **proposal** is to follow it, evaluating specifications. But the document addresses the definition of test bench, injection of signals, definitions for signal form generation, record and synchronization of signal values and even scripting, everything oriented towards interoperability of hardware to make tests and script reuse.
If not following a standard, the requirements must specify what (self)-diagnosis procedures are implemented and how to call them. For FMI, at least validation of machine state as found in the model exchange standard should be carried out.

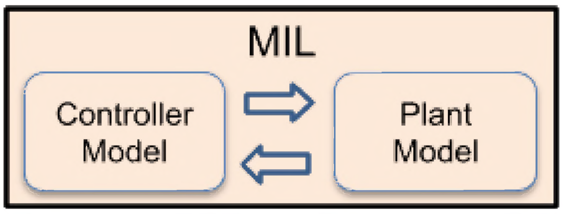
BBs have self-reflection. This is a very important point regarding validation. The validation for a platform must discover any component and ask for its description. This description refers to its internal structure, state… in particular and very important, the state of the BB can be interrogated. FMI for model exchange defines (ch 3.2.3) the state machine for calling sequence of C functions.
FMUs can have parameters and tunable parameters. While a BB can define its own variables, the inputs, outputs, parameters… will be exported in deployment to external tools. The proposal is that layer 3 offers an OPC UA server to access BBs' data. Requirements should be written that define which signals are exported and which companion standard is used. The supported companion standards names will be included in the description file. Using of several namespaces is possible in OPC UA. For instance, the position of an axis could be named with the sercos companion standard and, at the same time, accessed with the VDW companion

standard for machine tools. Verification will assure that named variables do exist and can be accessed.

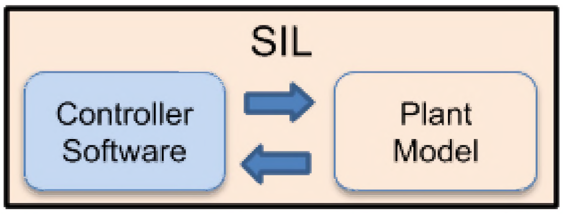
BBs maintainability is of no relevance for testing except where there is an exposed information or behaviour that should specifically be tested, as is the case of BB3, where this functionality is covered.

5.4. *I-MECH Model Based Design*

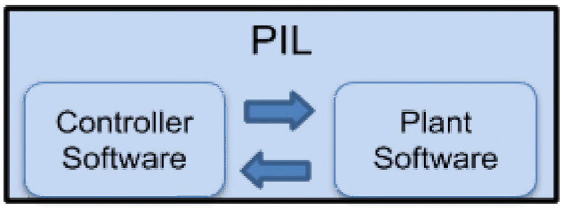
A very short description of the different XiL phases follows. It is just an introductory part to fix definitions and controlling context. HiL is usually not very well understood or is mixed with rapid prototyping or even deployment. In fact, there are many possibilities regarding HIL hardware. In what follows, background color is orange for simulation hardware and blue for target hardware or software.



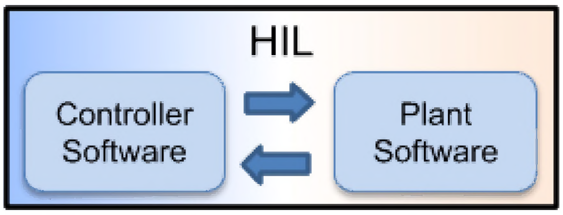
MiL is the only mandatory phase for Model Based Design, everything is in the simulation platform. Depending on availability and designer needs, plant model can vary from functional models to detailed systems models where every relevant phenomena, including non-linear effects, field buses and noise could be captured. Depending the level of details and computational power, simulations speeds could be faster to slower than real time.



In SiL, all software is compiled (no more dependency on simulation tool) and simulating with fixed step solver at final sample rate, but not in real-time yet. There are no timing issues, as time is simulated, and complex models are then possible. The aim of SIL is to verify that the code works as expected in simulation, that is, that the controller model and the controller software have similar behaviour.
The blue color means that the code for the controller software is compiled while running (orange color of the enclosing rectangle) in a simulation platform.



In PIL phase, *controller code runs native on the definitive platform* (that's why the color of blocks is blue). The definitive platform could be emulated. The aim is to verify that the code works as expected in real time and also interaction with other software parts, interrupt system, etc…



In HIL phase, *controller code runs native on the definitive platform and plant model or software runs in a **different** platform*.(Hence the two background colours)
The plant can be implemented in hardware, FPGA, industrial PC...or be directly the Simulink model running in the simulation platform. There are even dedicated hardware platforms to download directly models from Simulink. The aim is to validate the code on complex models without the real plant.
Rapid prototyping is often confused with HIL, while being different. Rapid prototyping needs first a prototype or even the real mechanical device and analog hardware (for instance a motor and a driver or class D amplifier). A platform where control software coming from the controller model and some real time software drives the prototype and is used to validate algorithms. This platform can be the one provided by BB10 and BB11.

I-MECH architecture defines three layers for control applications. Depending on the XIL phase these layers fall in the left or right part of the above figures. For every specific BB the mapping of layers to processors must be defined, but not all the phases must be implemented (some are not even possible).

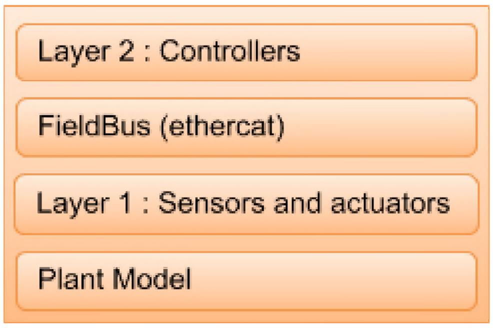
5.5. I-MECH Model in the Loop (MiL) V&V Methodology

For Model based Design this step is mandatory.
As identified in D6.2, Simulink is the de-facto standard for development and validation in MIL, however there are other simulation tools available which should be also considered, especially those which are based on open standards. Therefore I-MECH BBs must take into account different scenarios for MIL.

- In case of Simulink-based development, the BB shall be published as Simulink library (for use in Simulink) and also as FMU (for import in other simulation tools).
- In case of BB developed in other tools, it shall be published as a FMU (for import in Simulink).
- In case of BB developed in C/C++, it shall be published as an S-Function (for use in Simulink) and also as a FMU (for import in other tools).

Dealing with all these scenarios the interoperability of I-MECH BBs between simulation tools in MIL is guaranteed.
It should be remarked that since latest Matlab release, i.e. R2018b, there is FMU Import natively supported and FMU Export (Co-simulation mode) natively supported via “Tool-Coupling

Co-Simulation FMU Export for Simulink", although end user still needs Simulink license. Another alternative for FMU Export from Simulink can be the Modelon tools.



The FMU file contains everything to simulate the BB in a simulation platform that can import this file format (typically windows or linux, on 32 or 64 bits base). There is standard a tool (FMI compliance checker) that verifies that such an FMU is compliant with FMI requirements. Using FMI for MiL couplings requires to be careful (as any kind of fixed-step continuous simulation couplings) regarding the coherency of FMU time steps with system dynamics and controllers resolutions. Timesteps larger than critical systems dynamics will lead to simulation incoherencies, which are diagnosticable by studying steady state oscillation (pumping), or high dependency of simulation results from simulation steps. Algebraic loops could also lead to cosimulation issues, but could be broken with usual system simulation strategies. Even if both FMI for co-simulation and model exchange are available, the co-simulation one should be preferred as it's the most behaviorally conservative.

I-MECH BBs must have written requirements for functionality (they already have them in D4.2 and D5.2 documents). If adhering to ASAM-XIL or with a similar system, the functional requirements could be **tested** in the simulation platform.

Regarding I-MECH layer distribution, all the interfaces and used BBs are included in the modelling platform. For L2-L3 interface, tuning and/or monitoring scripts should be provided to evaluate the functional requirements of the BB, as proposed in Appendix B. L1-L2 interface (typically the Ethercat fieldbus) can be modelled or not depending on design engineer needs.

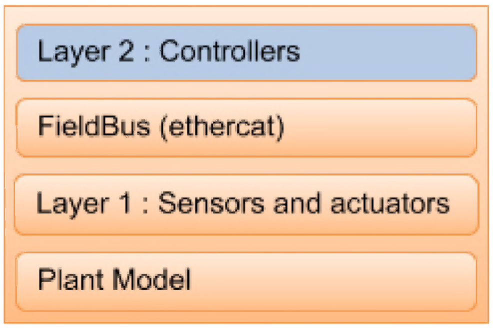
User interface at this level is provided by the simulation platform.

Plant Models can be done directly in the simulation platform language or imported. The preferred format is .fmu files with, may be, some extensions. Verification and Validation must be done at least at this level.

5.6. I-MECH Software in the Loop (SiL) Test Methodology

FMUs have descriptions for the platforms supported by the code. If one of these platforms is similar to the simulation platform or there's an emulator for it, SiL can be carried out for the control BBs. (BB6-BB9). For instance, if x86 is supported and the target for deployment is also an x86, object code (may be a dll or so) can be used. Potential license issues have also to be considered, as most of the simulation software still call for licenses systems with FMIs (unless they have the option to remove the license calls). FMI specific overheads also have to be considered. Another way to implement MiL is also to directly implement part of the code to be

verified as a simulator component. This approach is only applicable with low-level programs typically in C, with limited specific dependencies.
Only functional validation makes sense here. Verification has been carried out at MiL. But here one can validate that the chosen step size is correct or even measure control loop time (some emulation targets count instruction cycles, for instance).

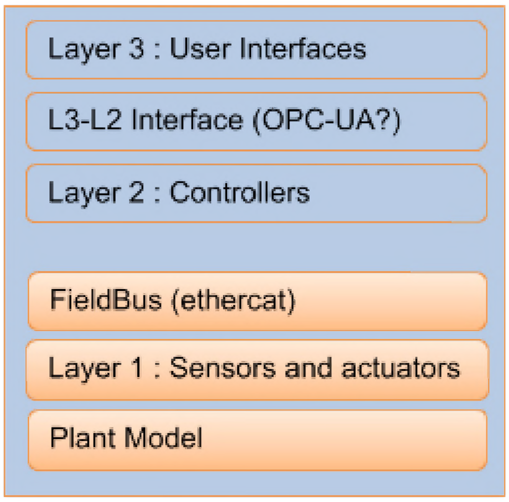


Thanks to this phase and the V methodology, early detection of otherwise complex to find problems can be detected (for instance, longer than expected cycle times due to events, etc…)
Regarding I-MECH layer distribution, there aren't many changes, and everything is performed in the simulation platform.
User interface is done with the simulation platform means.

5.7. I-MECH Processor in the Loop (PiL) Test Methodology

This phase can be reached from MiL directly or from SiL. The final platform must be available.
Two approaches are possible on this phase:

- The first is to fully virtualize the control platform (OVP, qBox, Imperas, Veloce, SIMIT…), so reproducing the full SW and HW behavior based on the final design. These virtual platforms could interact with the plant models to be operated through dedicated interfaces (SystemC, FMI) or network (OPC UA)
- The second is to load the model on the final target platform. In this case accuracy of the plant is no issue, but speed is important. Therefore plant models must be simplified when they are computational too complex.

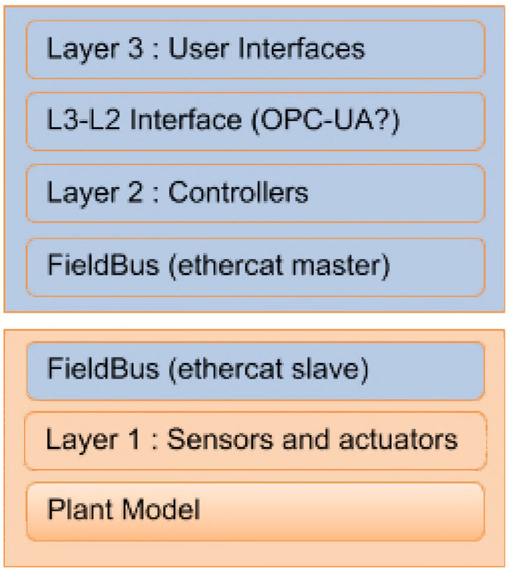

Layer 3 : User Interfaces
L3-L2 Interface (OPC-UA?)
Layer 2 : Controllers
FieldBus (ethercat)
Layer 1 : Sensors and actuators
Plant Model

Even with these restrictions, this is a very interesting phase where errors due to interaction of the control loops with the general timing and software of the target platform can arise and are easier to correct (and less dangerous) than in the real prototype.
As in the SiL case, I-MECH verification done in MiL is valid and what this phase adds is validation in the real target. As the models will be possibly simplified, this phase doesn't eliminate the need for further functional testing or validation (for instance, friction would not typically be considered at this level and must be carried out later).

One specific step in this engineering phase for I-MECH is that layer 3 should be fully functional with respect to protocols (e.g. OPC UA) and that the information models (e.g. OPC UA namespaces and companion standards) can be validated in this or in the HIL phase.
This introduces as requirement that the companion standard to be used for a BB must be declared somewhere. It should be included as a new tag in the FMU's xml file.
PIL always has a simple plant model, fitting within the computational constraints of the target platform, but it is anyway a good point to verify many of the controller functional requirements under real conditions. With simple plant and layer 1 models, this is a very useful phase to validate higher order requirements or Building Blocks, like trajectory generators, etc…
Moreover, many of the functionalities of the layer 3 can be verified and validated here, for instance, OPC UA protocol, control variables and parameters discovering and read-write, etc…

5.8. I-MECH Hardware in the Loop (HiL) Test Methodology

The HiL phase is very interesting for problems where the plant is complex, big, expensive or just unavailable… and a detailed model, while also costly to build and validate, is the best solution to try different control strategies. These could fire errors that would damage the machine if tested on a prototype.
Here, the control loops will run on the target platform which is interfaced with RT plant models. There's more than one possibility to connect the target to the plant simulator, but the preferred way, when possible, is using in the target all the final hardware (for instance the field buses) and have its counterpart in the RT simulation platform (dSPACE, NI, or PC with interfaces conditioning, common buses...). Hybrid systems exist where some hardware (sensors or actuators) are also used.

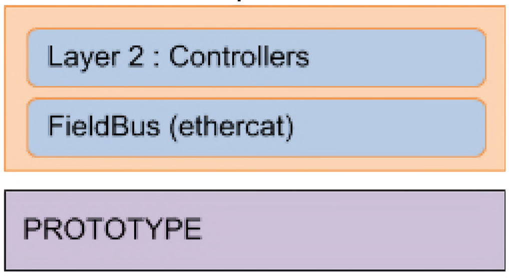

Layer 3 : User Interfaces
L3-L2 Interface (OPC-UA?)
Layer 2 : Controllers
FieldBus (ethercat master)
FieldBus (ethercat slave)
Layer 1 : Sensors and actuators
Plant Model

HiL, while requiring investment in terms of equipment and RT simulators building and validation, can be mandatory for highly complex systems or where machine availability is scarce and the technical risk is high. As physical models come from different modelling tools and simulation platforms are available from various vendors, the same requirement in I-MECH methodology is mandatory. To work in an I-MECH ecosystem, the plant must be modelled as an FMU RT or Simulink plant model.
In case of FMU RT, this step can be a little bit more complicated in that, for some models and tools, co-simulation is needed. This has already been addressed in the FMI specification, and FMUs can have the files that the simulation platform uses. The description of environments, supported information

models, etc… is written in an xml file inside the .fmu file (a zipped directory).
HIL determines that the target platform is the final platform, so that both control layer (layer 2) and interface layer (layer 3) are in its final version.
Depending on the quality of the models for sensors and plant, everything can be verified and validated, including data gathering and high level algorithms.
For instance, even temperature models can be run on the simulation platform or failure injection can be done to trigger error procedures in the controller.
HIL may have drawbacks if complex models for sensors and plants are needed. They are expensive to build and validate, and a high-end processor is needed. For very detailed simulation, models in different formats could be needed (e.g. p-spice models of power electronics to care for effects of dead times and other non-linearities) and a HF test bench can be cheaper while not as flexible. Opportunities like selectively reducing models (linearized, RSM networks, neural networks, etc…) could solve RT compliance issues by enabling a good performance-representativity tradeoff.

5.9. I-MECH Rapid Prototyping (RP) Test Methodology

This is a technique used by some I-MECH project partners for validation and test of their systems.
It is useful when there's a mechanical prototype available (example, a motor and amplifier…) with inputs and outputs.
The control system runs on a platform where direct downloading of control software from the simulation tool is possible. User interface and access to signals, etc… are available as layer 3 software (for PIL, HIL and RP). The whole system greatly simplifies development of control algorithms, especially for "small" systems where the prototype can be directly the plant or a simplified, scaled, mock-up of it. Design of the controller is done graphically like in simulation and many high level blocks are ready for signal processing, etc... National instruments offer such systems, as well as some companies partner Simulink for the same purpose (dSpace, Speedgoat), the same companies behind HIL dedicated hardware.

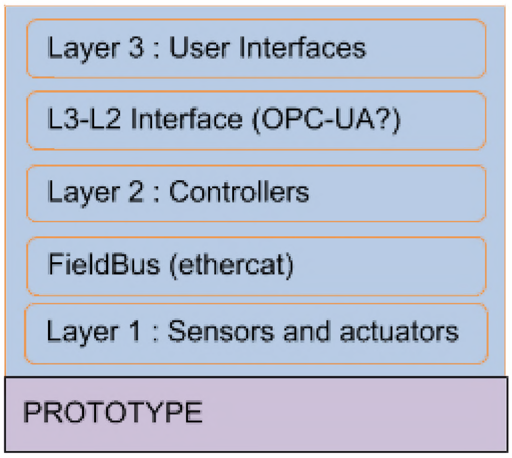


Functional requirements verification can be done as the dedicated platform allows injection of signals and use of many of the verification and validation blocks of the native simulation platform.

5.10.I-MECH Deployment Test Methodology

HIL mode or even rapid prototyping is very close to the reality. But, regarding verification and validation of I-MECH BBs, the procedure can be carried on an individual basis, BB by BB with specific models for functional validation of key features. When combining several BBs and other legacy control code or general software, some problems can happen and new behaviours must be verified.

When dealing with a bunch of BBs and connections between them, new problems regarding calling order, resolution of possible algebraic loops, etc…must be taken into account. To solve these problems, an "FMI master" will be an important part of the system, or a scheduler for Simulink generated software. When all the control code is generated by the simulation system, this is left to its internal algorithms. But when dealing with a hybrid system of new and legacy code some rules must be defined or the legacy code must be imported somehow to the simulation platform.


Layer 3 : User Interfaces
L3-L2 Interface (OPC-UA?)
Layer 2 : Controllers
FieldBus (ethercat)
Layer 1 : Sensors and actuators
PROTOTYPE

Finally, regarding connection hardware, some virtualization means are necessary if some degree of "plug and produce" or whatever plug & play strategy is needed. Ideally, replacing Ethercat with Profinet IRT or Sercos3 in a platform for a new project shouldn't need changes in the I-MECH BBs or its combinations.

All these issues must be considered in the BBs architecture, especially in BB10-BB11.

5.11. Hardware and software integration

The validation and verification for integration the hardware and software building blocks, focus on the collaboration of the building blocks. This is shown in three areas: interfacing, methodology and performance.

- Interfaces:
  - Can building blocks communicate with the standardized external interfaces (layer 1 -> layer 2) and (layer 2 -> layer 3)
  - Do building blocks on different layers find each other and work together when needed for the functionality of the building block?
- Methodology:
  - Are software modules exchangeable across hardware platforms?
  - Are HIL (hardware in the loop) tests supported?
- Performance:

- Are performance requirements met?
- Is separation of concern reached, in a sense that one performance wise non behaving building block does not affect the performance of another building block?

The software hardware Integration will take place in separate phases.

- a pre-integration phase
- Integration the software blocks on the target hardware platform
  o Focus on layer 2 parts of BB6, BB7, BB8, BB9 with BB10/BB11
- Integration of sensors and actuators on layer 1 with the target hardware platform
  o Focus on EtherCat communication to layer 2.

The pre-integration step will bring software building blocks on layer two together in the simulation environment (i.e. matlab simulink) and will show collaboration on the development default hardware target. This enables the Model based designs as much as possible.

The target platform integration will bring the layer two building blocks to the two reference target platforms. Finally the sensors and actuators on layer 1 will be integrated with the I-MECH system.

6. I-MECH V&V definition for Building Blocks

BBs (and platforms) have a service oriented architecture where the already defined states and behaviours can be explored and forced somehow. This is again needed for validation.
This is the case of FMI for Model Exchange (chapter 3.2.3 of specification). At least **verification** can be done with the FMI compliance checker.
The I-MECH platform layer 2 must be addressed by BB10/BB11. Defined services (for instance Fieldbus virtualization or scheduling) must be implemented and made available for control BBs. It is not required that the complete BB10/BB11 is modeled as a MIL simulation, but it is necessary, at least, a simple behaviour of this interface.

Validation refers to assess that the BB does what it is expected to do, what has been written in the upper level requirements, that is, refers to behavioural aspects. From this point of view, D4.2, D5.2 are valid (validated) requirements for BBs.
If further steps on validation are needed, for instance recording output and state for defined input signals, that task can be also view as testing. In any case, that test or validation will be done in the simulation platform.
If compatibility and interoperability of tests, input signal definition and scripts is required, the preferred standard would be ASAM AE XIL Simulation Mode Access. But at the time of this

writing we have no evidence of compatibility with standard commercial tools like Simulink. In that case this validation task should be made using programming scripts.

The proposal is that only MIL is mandatory, so that validation will be done in that mode. Validation means that behavioural requirements are written. For example, having a BB that implements a PI with anti-windup and saturation, these features must be validated.
For specific BBs, validation can be done also in SIL, PIL, HIL… As explained in the previous chapter, the objectives are different in each case. For instance, PIL can find platform-specific software defects (object code depends also on compiler) and potential problems related to real-time execution of control algorithms and interrupt handling including jitter and resource corruption. SIL, on the other side, can be a better tool to benchmark BB's performance against different models or model parameters using the final source code. But it seems it is not possible to set these validations as mandatory. This could be left to BBs owners' decision.

In what follows, and before describing it in detail, a mental model of the BB represents a function relevant for a mechatronic platform that has an FMU file and can, then, be simulated in MIL mode, what is mandatory. The FMU for Model Exchange has a state machine driven by "C" function calls. This state machine should somehow maintained for deployment, at least when code generation is done automatically. Some mapping between these calls and field bus initialization phases is needed. Some configuration tasks are usually available only in some phases (or states) and transitions are commanded in a documented way. Standards are applicable to all these phases.
In the I-MECH methodology we should describe an abstraction for such phases in such a way that initialization can be tied to other BB events (like integrator initialization). A different, maybe easier approach, is defining a state machine for I-MECH where those initializations can be done. An example of a simple BB will be provided as a guide to explore the mentioned concepts.

One possible set of initial architectural requirements regarding this model can be:

Requirements for Modularity and Composition
rq-D6.1-ARCH: a complex BB that is composed of inner BBs connected must comply with I-MECH requirements and objectives. (Refer to FMI, FMU and related MA project : "System Structure and Parameterization")

Requirements for self-diagnosis
rq-D6.1-ARCH.sd.1
Every BB must have an interface which let injection and test points. This interface is proposed to be adhered to ASAM.
rq-D6.1-ARCH.sd.2

Every BB must provide a set of services to evaluate if the requirements are met.

Requirements for self-reflection

rq-D6.1-ARCH.sr.1
Every BB must have at least an FMU file with the FMI for Model Exchange part filled.
rq-D6.1-ARCH.sr.2
I-MECH BBs must provide information on its function, behaviour, intended use…in the FMU xml description file.
rq-D6.1-ARCH.sr.3
I-MECH BBs must implement an OPC UA server or provide information to access the inputs, outputs, states and tunable parameters from a script.
rq-D6.1-ARCH.sr.4
For OPC UA, I-MECH BBs must define for deployment the companion standard that defines the semantics of its variables.
rq-D6.1-ARCH.sr.5
BB owner will choose the companion standard for I-MECH BB semantincs. Verification will test that variables exist and can be accessed. The list of available companion standards can be found in the following link:
https://opcfoundation.org/developer-tools/specifications-opc-ua-collaborations

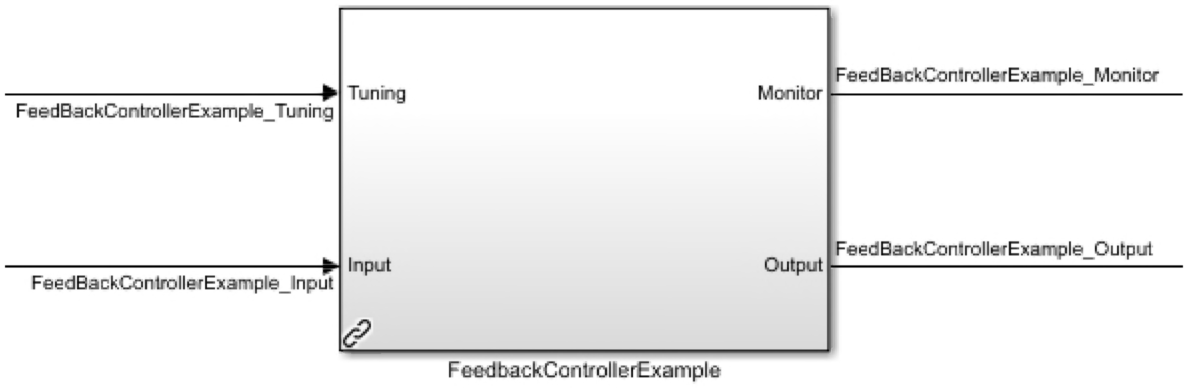
Requirements for Interoperability

rq-D6.1-ARCH.io.1
I-MECH BBs must have a Functional Mockup as defined in FMI and possibly also Simulink and/or S-Function implementation.
rq-D6.1-ARCH.io.2
At least MIL mode must be available for that BB in simulation platforms that can import FMUs. This means that at least the Model Exchange part is available.
rq-D6.1-ARCH.io.3
At least code for a platform must be available.
rq-D6.1-ARCH.io.4
An I-MECH BB must have input, output, parameters and variable signals. The type of the signals must conform to FMI specification 2.0.

7. I-MECH V&V for Software Building Blocks

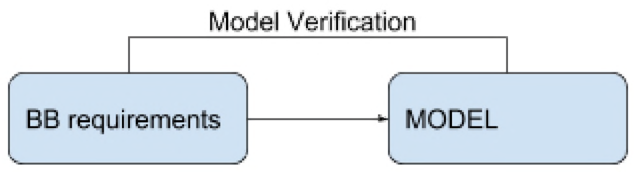
Any I-MECH compatible Building Block must comply with the objectives of the I-MECH architecture. These objectives are dealt with also in D2.4 for a reference platform, where also FMI standard is mentioned as defined in WP6 meeting in Eindhoven. As this WP6 addresses verification of BBs, a detailed example of how a BB could be implemented that complies with the requirements will be helpful.
D4.1, Motion control requirements and specification (first iteration) defines specifications for a control system in chapter 4. Moreover, **discrete control is assumed** in all of them, as defined in chapter 2.8.

Those requirements are functional requirements. It is assumed that general architecture requirements have been already verified (there is a FMU…)
One important point and requirement that must be validated is that any parameter can be accessed for reading or configuring from layer 3 upwards with OPC UA or any other method using programming scripts, and also written if it is a tunable parameter. See “Tuning” and “Monitor” in the figure below:


FeedBackControllerExample_Tuning
Tuning
FeedBackControllerExample_Input
Input
Monitor
FeedBackControllerExample_Monitor
Output
FeedBackControllerExample_Output
FeedbackControllerExample

For these control requirements, something more specific must be written if it has to be validated. Even if not validated in the same platform, this document is about methodology and benchmarking, what means that something must be measured.
For every requirement the specific BB should specify target figures and method to test them and how that BB improves state of the art. This takes part in the benchmarking expected.

Guideline for requirements of software BBs.
At this specific stage of Model Based Design, the software BBs must be validated **at least** in MIL mode.



Functional requirements of BBs are gathered in a file of xml format.

All functional requirements must follow the naming convention:
rq-BBx-FUN.yyy.zz
where x is building block, yyy are text and zz number

The requirement verification method and obligation is also indicated.

T:	test/validate
I:	inspect/demonstrate

where a requirement can be:

R:	required (must-have)
O:	optional (nice-to-have)

A must-have functional requirement can be demoted to a nice-to-have verification requirement if its principle is conceptually demonstrated elsewhere, according to the verification strategy.

For every functional requirement at least the following information must be provided:

<VerificationMode></VerificationMode>	: MIL,SIL,PIL,HIL,Deployment
<ID></ID>	: the name in the specified convention
<Description></Description>	: text explaining the requirement.
<Reason></Reason>	: why do we require this, intended purpose
<Verification></Verification>	: How to verify/validate it

<Inputs></Inputs><Results></Results> (optional when a value different from pass/don't pass is expected or wanted, example, rejection of input noise > 30dB for a test input can be a true/false condition, but in some cases a vector test and the expected output value is needed)
The same requirement can have more than one input if verified in MIL and HIL, for instance. Hence, ID and VerificationMode fields are mandatory to uniquely identify a test.

This verification on the document can be done very easily via a scripting language.

Regarding verification results, the link between files is done by ID and VerificationMode..
The results file can be again be an xml file.

For every requirement and mode, the following information must be provided.
<VerificationMode></VerificationMode> : MIL,SIL,PIL,HIL,Deployment
<ID></ID> : the name in the specified convention
<Result></Result> : success, error
<Inputs></Inputs><Outputs></Outputs><Values></Values>
To arrange numeric results of tests.

8. I-MECH V&V for Hardware Building Blocks

Explicitly the hardware BBs under development in I-MECH are as follows:
BB-1 Platform for Smart Sensors with Advanced Data Processing
BB-2 Real-time wireless sensors providing complementary feedback information
BB-3 Robust condition monitoring and prediction diagnostics
BB-4 High Speed Vision
BB-5 High performance servo amplifier design
BB-10 Development / selection of control specific multi-many core platform

There is a requirement to validate the I-MECH BBs as a minimum requirement in MIL mode. For hardware BBs where the objective is to move directly to HIL test and validation this supersedes the MIL validation requirement i.e. where HIL validation is conducted there is no requirement to implement MIL validation. It is also possible to validate the hardware BB component as a rapid prototype (RP). The test methodologies associated with MiL, HiL and RP test are outlined in section 5 of this document.

Guideline for determining requirements to test under WP6 for Hardware BBs.
As stated previously an I-MECH compatible Building Block must adhere to the objectives of the I-MECH architecture. These objectives are outlined in D2.4. The functional requirements for the I-MECH hardware building block components will be validated against the requirements developed in WP2 and further refined in WP3 in Tasks 3.1 and 3.2. System behaviour layer integration and connectivity requirements and specification for each of the components are sources in WP5 in Tasks 5.1 and 5.2. Where each requirement will be validated in the I-MECH work program is illustrated in figure 4. As part of establishing which requirements will be tested against in WP6 the following information will be compiled as part of Task 6.3:

- Each BB owner will examine the currently compiled list of functional requirements for their respective BB.
- The BB owners will map the functional requirements to the generic I-MECH requirements presented in D2.4 and in D6.1 under each of the following headings:

modelling test methodology XiL, interoperability, modularity, maintainability, self reflection and self diagnostics.

- Information on how each identified requirement will be validated will be presented. The proposal is that the requirement verification method and obligation is also indicated:

T:	test/validate
I:	inspect/demonstrate

where a requirement can be:

R:	required (must-have)
O:	optional (nice-to-have)

- Where there is no associated requirement under a given heading, information on how this condition can/will be addressed in future will be provided.

Guideline for validation test under WP6 for Hardware BBs.

The test specification template presented as Annex A in this document will be employed to compile and disseminate the validation findings for each of the BB hardware components. As per the template the test package specification will contain information on the following:

- A functional description of each test case shall include the test purpose, setup, functional requirements and acceptance criteria.
- Information on test operation including user interface functionality, operation modes/instructions and test log file descriptions.
- Listing of functions and failure modes to be tested and corresponding test cases.

Acknowledgement

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737453. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and Netherlands, Czech Republic, Latvia, Spain, Greece, Portugal, Belgium, Italy, France, Ireland.

Appendix A: I-MECH Building Block Test Documentation - Template

This section describes the I-MECH Building Block Test Documentation that will be employed as part of the verification and validation.

I-MECH Building Block Test Package Specification

I-MECH Building Block Test Package (to be developed before conducting validation/test)
Based upon the functional description, each test case shall be described specifying (where applicable): ● Test purpose ● Test setup including overview of building block component hardware, software and functions to be tested (hardware and software components, parts, serial numbers and software versions) ● Specification or reference to I-MECH building block requirements and functional requirements ● Expected results and acceptance criteria: test/validate/inspect/demonstrate
Description of I-MECH Building Block test operation, providing information on (where applicable): ● User interface functionality ● Operation modes of Building Block test simulator ● Operating instructions, and ● Presentation of trends and test log-files e.g. FMI compliance
Listing of functions and failure modes to be tested and corresponding test cases (where applicable). Failures to be simulated can include but not limited to: ● Sensors or input devices failure modes (dropout, noise, calibration, errors, drift, bias, ….) ● Failure mode of actuators, drives, power system components ● Failure mode of electro-mechanical components ● Feedback from sensors on actuator failure modes ● Failure modes in computer networks

Other information to be included in test package documentation (where applicable): ● List of involved companies, name of contact person/title, contact information ● Test activity schedule ● Risk assessment of planned testing.

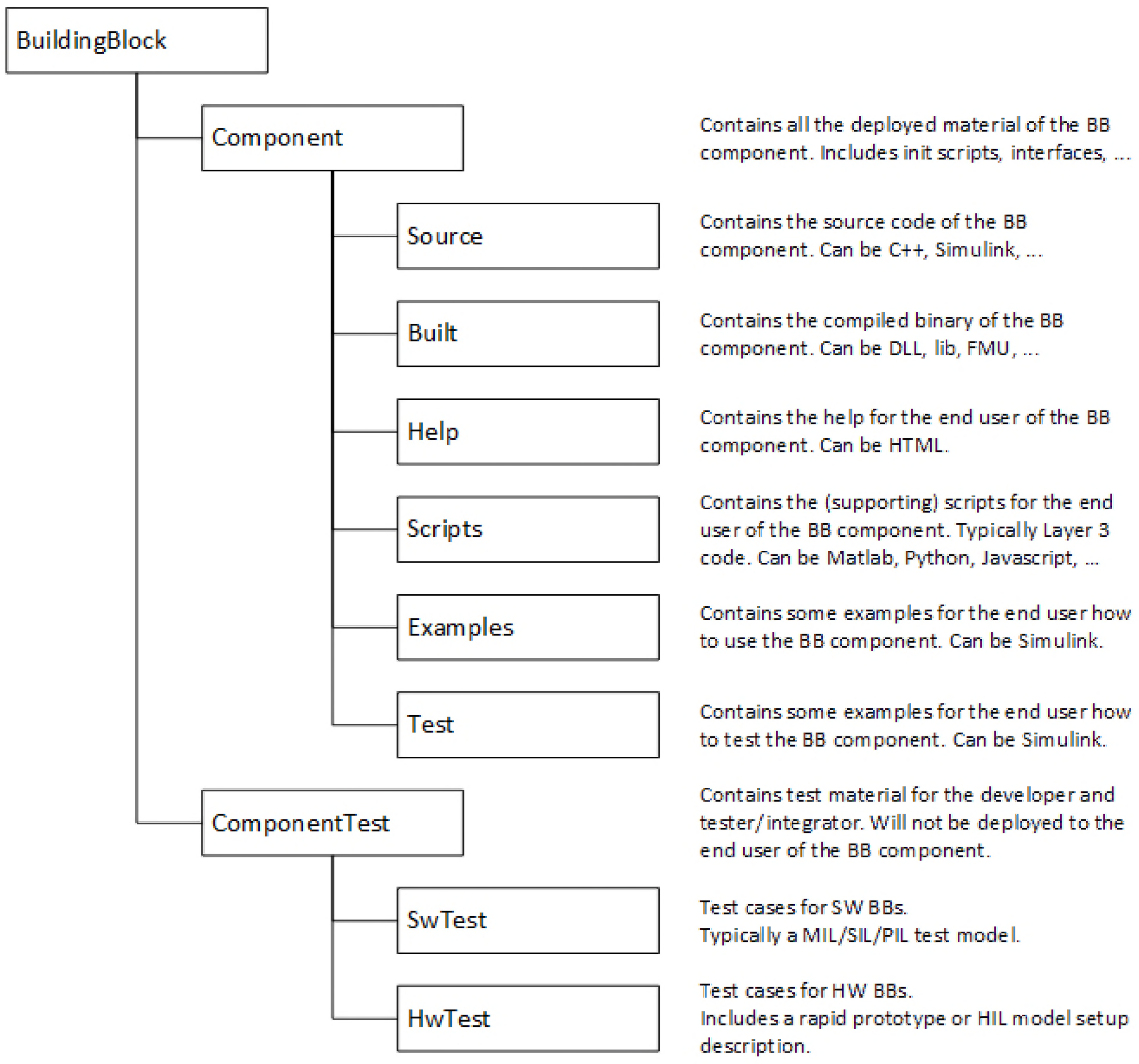
I-MECH Building Block Test Report Specification

I-MECH Building Block Test Report (to be developed following conducting validation/test)
Test reports to include the following information (where applicable): ● Listing of functions and failure modes test and corresponding test cases ● Recorded results of each test case. ● Description of each finding from functional requirements as observed in any test case recorded with sufficient detail to be followed up further. ● Any functions incomplete or not available for testing shall be recorded as a finding.
Other information to be included in test report (where applicable): ● List of involved companies, name of contact person/title, contact information ● Link to trends and test log-files output from test cases

Appendix B: I-MECH Building Block Structure - Template

This section describes an example of a structure of a BB deliverable. This example serves as a suggestion, not as a mandatory structure, since each BB can have unique characteristics.
The main objective of this example is to provide some guidance how to distribute a BB to the end users and to the V&V integrators/testers.

The example BB directory structure is shown in the figure below:



Contains all the deployed material of the BB component. Includes init scripts, interfaces, ...

Contains the source code of the BB component. Can be C++, Simulink, ...

Contains the compiled binary of the BB component. Can be DLL, lib, FMU, ...

Contains the help for the end user of the BB component. Can be HTML.

Contains the (supporting) scripts for the end user of the BB component. Typically Layer 3 code. Can be Matlab, Python, Javascript, ...

Contains some examples for the end user how to use the BB component. Can be Simulink.

Contains some examples for the end user how to test the BB component. Can be Simulink.

Contains test material for the developer and tester/integrator. Will not be deployed to the end user of the BB component.

Test cases for SW BBs.
Typically a MIL/SIL/PIL test model.

Test cases for HW BBs.
Includes a rapid prototype or HIL model setup description.

Notes:

- The "end user" is the developer of the customized application (e.g. an I-MECH pilot). The end user receives the BBs from the BB owners and constructs a controller with them.
  The "end customer" will be the final user of the controller received from the end user.
- A BB owner may deliver multiple (independent?) components, thus resulting in multiple directories. Rename the <Component> directories accordingly.
- At least one of the <Source> or <Built> directories is mandatory, the other may be omitted (e.g. when the source code is not deployed).
- The <Help> directory provides user instructions. E.g. available from the Help function of a Simulink block (which displays a HTML page).
- The <Scripts> directory contains various scripts and functions that assist the end user. The end user may deploy these scripts (possibly modified and/or compiled) to the end customer. The BB owner could make a clear distinction between developer scripts (for end user) and layer 3 scripts (for end customer).
- An open issue is how to provide the data dictionary for OPC UA and how to test a BB with OPC UA.
- The <Examples> and <Test> directories are optional, and provide further models and information to the end user about the correct usage of the BB component.
- The <ComponentTest> directory is mandatory, but won't be delivered to the end user. This directory is of interest to the BB owner (BB development testing) but also to the WP6 tasks that V&V the BB components as described in this document
- A BB owner may provide multiple test cases (e.g. for the different use scenarios or for the different BB10 platforms, CompSOC and x86 COTS). Rename the <ComponentTest> directories accordingly.
- The <SwTest> directory is mandatory for SW BBs. It contains a MIL test model (with supporting scripts) that V&V the BB requirements. It also contains a PIL test model (with Layer 3 scripts) that demonstrate the deployment of the BB on both BB10/11 platforms.
- The <HwTest> directory is mandatory for HW BBs. This can be either a HIL test model or a rapid prototype description, defined by the BB owner, to V&V the BB requirements. Preferably using the BB10/11 platforms, but the issue is that the HAL interface (particularly access to the EtherCAT master) is not defined yet.

Sioux CCM will provide a template model of a generic feedback controller (as suggested for BB6) that complies to this template and provides more suggestions e.g. naming conventions. This template is explained in presentation 18120601R01 located in the partner zone.