Grant Agreement: 872698

Digital innovation HUBs and CollAborative Platform for cyber-physical systems

Third Version of HUBCAP Contents

Technical Note Number: D6.4
Version: 1.0
Date: September 2021
Public Document

https://www.hubcap.eu/

Editors

Stefano Tonetta (FBK)

Contributors

Stefano Tonetta (FBK)
Pietro Braghieri (FBK)
Alberto Griggio (FBK)
Nur Alam Labu (FBK)
Román Felipe Bastidas Santacruz (POLIMI)

Reviewers

Adrian Pop (RISE)
Adeel Asghar (RISE)

Consortium

Aarhus University	AU	Newcastle University	UNEW
Fortiss GmbH	FOR	Virtual Vehicle Research GmbH	VV
Fondazione Bruno Kessler	FBK	KTH Royal Institute of Technology	KTH
University “Lucian Blaga” of Sibiu	ULBS	Engineering Ingegneria Informatica S.p.A.	ENGIT
Research Institutes of Sweden AB	RISE	F6S Network Limited	F6S
Politecnico di Milano	POLIMI	Unparallel Innovation	UNP
Controllab Products	CLP	BEIA Consult	BEIA
Verified Systems International GmbH	VSI	Validas	VAL
Technology Transfer Systems srl	TTS		

Document History

Ver.	Date	Author	Description
0.1	24-07-2021	Stefano Tonetta	Initial structure
0.2	24-07-2021	Pietro Braghieri	Added Chapter on OSLC-based interaction
0.3	23-08-2021	Stefano Tonetta, Alberto Griggio, Román Felipe Bastidas Santacruz	Added contents to Chapters on certifying model checking and TRL arguments
0.4	24-08-2021	Nur Alam Labu	Added contents to chapters on summary of current HUBCAP contents and Experiment within a sandbox.
0.5	29-08-2021	Adrian Pop	Added diversification of analysis chapter
0.6	30-08-2021	Stefano Tonetta	Final revision before internal review
0.7	20-09-2021	Nur Alam Labu	Revision addressing internal reviewers' comments
1.0	30-09-2021	Stefano Tonetta	Final editing

Abstract

The **HUBCAP Collaborative Platform** will provide **catalogues of models and tools** for trial experiments with Model-Based Design (MBD) technology to design and develop innovative Cyber-Physical Systems (CPS) solutions with the support of MBD.

The HUBCAP platform is already operational and contains tens of tools and more than a hundred models. The Task 6.4 of the project enhanced the platform with features focused on the validation of the underlying model-based tools or validation of its results: first, we demonstrate in the platform how it is possible to enrich the analysis results of some tools with proofs that can be checked by other tools; second, the tools catalogue of the collaboration platform has been improved with information and arguments in support of the definition of tools' Technology Readiness Level (TRL).

D6.4 is a document of **type "OTHER"**, given by the actual enhancement of the platform's contents. These can be browsed by accessing the platform. This document summarizes T6.4 contributions.

List of Abbreviations

API	Application Programming Interface
CCA	Common Cause Analysis
CLI	Command Line Interface
CMS	Catalogues Management System
COE	Co-simulation Orchestration Engine
CPU	Central Processing Unit
DIH	Digital Innovation Hub
DMA	Direct Memory Access
DSE	Design Space Exploration
FMEA	Failure Modes and Effects Analysis
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FTA	Fault Tree Analysis
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HSM	HUBCAP Sandboxing Middleware
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IdM	Identity Management
JSON	JavaScript Object Notation
KMS	Knowledge Management System
MBD	Model-Based Design
MBT	Model Based Testing
MCS	Minimal Cut Sets
OSLC	Open Services for Lifecycle Collaboration
RBAC	Role-Based Access Protocol
RCP	Rich Client Platform
REST	Representational State Transfer
SAT	Boolean Satisfiability Problem
SMT	SAT Modulo Theories
SME	Small and Medium-sized Enterprises
SSO	Single Sign On
SSP	System Structure and Parametrization
SysML	Systems Modeling Language
TFPG	Timed Failure Propagation Graphs
TRL	Technology Readiness Level
UML	Unified Modeling Language
UAV	Unmanned Aerial Vehicle
VDM	Vienna Development Method
VM	Virtual Machine
VNC	Virtual Network Computing

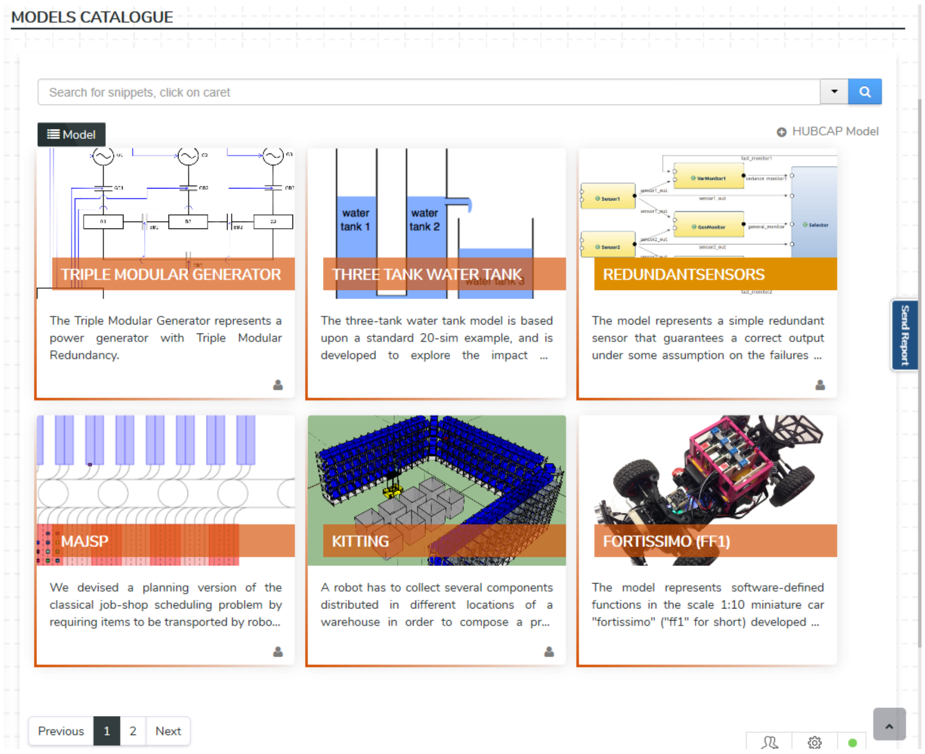
Contents

1. Introduction

Model-Based Design (MBD) is a method to address the design of complex systems with models. It prescribes the use of models throughout the development process in order to represent system structure and behaviours, providing a basis for machine-assisted analysis of system properties, and supporting design decisions through processes of refinement into implementation. The purpose is 1) to **reduce the complexity** of design **by abstraction**; 2) to **ensure the quality** of the system **by rigorous analysis** of its properties; 3) to **reduce the cost** of the development **by detecting issues** in the **early** phases of the development. MBD is quite standard in software engineering and is becoming more and more relevant in systems engineering, where it must integrate methods for control engineering and safety engineering.

MBD appears largely to be applied in domains such as aerospace where the return on investment can take decades. By contrast, SMEs require considerable flexibility to change processes to adopt MBD and may lack in-house expertise. In addition, the selection, procurement, training and deployment costs for some methods and tools can be discouragingly high. In general, it is difficult for SMEs to invest in acquiring the necessary background for example because of the high license fees from commercial vendors of MBD assets.

The **HUBCAP Collaborative Platform** will lower such barriers by providing **catalogues of models** and Model-Based Design (MBD) tools for trial experiments to design and develop innovative CPS solutions with the support of MBD technology. D6.4 consists of a third version of the platform's contents and, in particular, the enhancement for the validation of the tools' results. This document summarizes the contents of the catalogue for the reviewer's convenience.


MODELS CATALOGUE

Search for snippets, click on caret

Model

HUBCAP Model

TRIPLE MODULAR GENERATOR

The Triple Modular Generator represents a power generator with Triple Modular Redundancy.

THREE TANK WATER TANK

The three-tank water tank model is based upon a standard 20-sim example, and is developed to explore the impact ...

REDUNDANTSENSORS

The model represents a simple redundant sensor that guarantees a correct output under some assumption on the failures ...

MAJSP

We devised a planning version of the classical job-shop scheduling problem by requiring items to be transported by robo...

KITTING

A robot has to collect several components distributed in different locations of a warehouse in order to compose a pr...

FORTISSIMO (FF1)

The model represents software-defined functions in the scale 1:10 miniature car "fortissimo" ("ff1" for short) developed ...

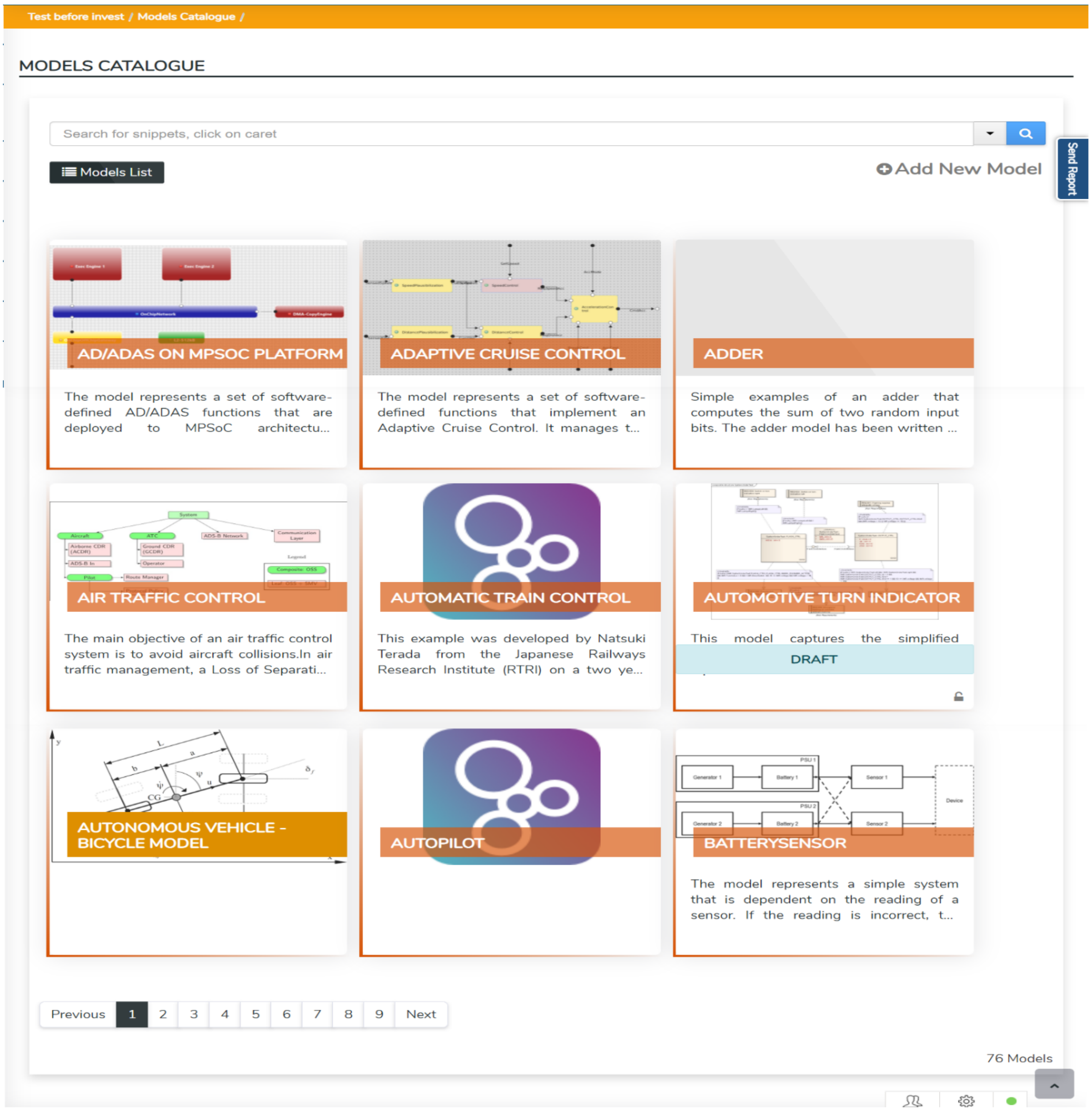
Figure 1: Screenshot of the models catalogue.

2. Summary of current HUBCAP contents

In the previous deliverables, we described the models and tools that were initially uploaded. These contents have been continuously updated, also contributed by the SMEs winners of the open call. The number of models in the platform is indeed contributing to the KPIs of the HUBCAP project.

We summarize here the current status of models and tools. The tool catalogue and the model catalogue are now also connected to the sandbox environment. It is able to launch the sandbox directly from the model catalogue with the "Try It" features.

The models cover various application domains including control engineering, electrical engineering, automotive, and avionics. The model catalogue currently has 76 models, almost all of which have been connected to the sandbox environment (Figure 2).


Test before invest / Models Catalogue /
MODELS CATALOGUE
Search for snippets, click on caret
Models List
Add New Model
Send Report
AD/ADAS ON MPSOC PLATFORM
ADAPTIVE CRUISE CONTROL
ADDER
AIR TRAFFIC CONTROL
AUTOMATIC TRAIN CONTROL
AUTOMOTIVE TURN INDICATOR
DRAFT
AUTONOMOUS VEHICLE - BICYCLE MODEL
AUTOPILOT
BATTERYSENSOR
Previous 1 2 3 4 5 6 7 8 9 Next
76 Models

The model represents a set of software-defined AD/ADAS functions that are deployed to MPSoC architectu...

The model represents a set of software-defined functions that implement an Adaptive Cruise Control. It manages t...

Simple examples of an adder that computes the sum of two random input bits. The adder model has been written ...

The main objective of an air traffic control system is to avoid aircraft collisions.In air traffic management, a Loss of Separati...

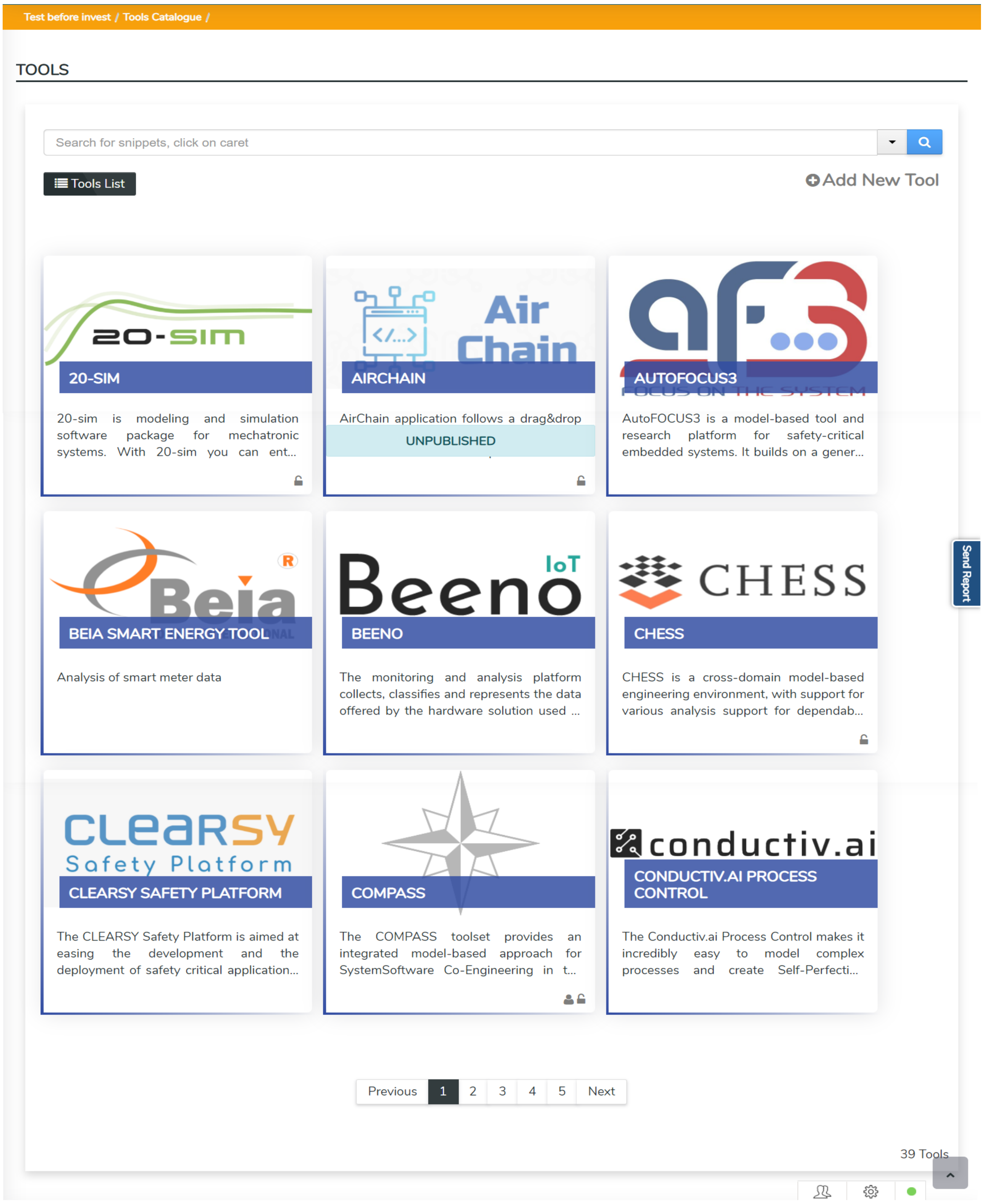
This example was developed by Natsuki Terada from the Japanese Railways Research Institute (RTRI) on a two ye...

This model captures the simplified

The model represents a simple system that is dependent on the reading of a sensor. If the reading is incorrect, t...

Figure 2: The Models catalogue.

At present, the tool catalogue has 39 tools and most of them have been connected to the sandbox environment (Figure 3). The sandbox can be launched directly from the tools catalogue with the "Try It Now" feature. These tools are provided by the HUBCAP partners, SMEs partners, and they are uploaded, installed, and tested inside the HUBCAP Collaborative Platform sandbox to constitute the contents of the catalogue of tools.


Test before invest / Tools Catalogue /
TOOLS
Search for snippets, click on caret
Tools List
Add New Tool
Send Report
Previous
1
2
3
4
5
Next

20-SIM

20-sim is modeling and simulation software package for mechatronic systems. With 20-sim you can ent...

AIRCHAIN

AirChain application follows a drag&drop

UNPUBLISHED

AUTOFOCUS3

AutoFOCUS3 is a model-based tool and research platform for safety-critical embedded systems. It builds on a gener...

BEIA SMART ENERGY TOOL

Analysis of smart meter data

BEENO

The monitoring and analysis platform collects, classifies and represents the data offered by the hardware solution used ...

CHESS

CHESS is a cross-domain model-based engineering environment, with support for various analysis support for dependab...

CLEARSY SAFETY PLATFORM

The CLEARSY Safety Platform is aimed at easing the development and the deployment of safety critical application...

COMPASS

The COMPASS toolset provides an integrated model-based approach for SystemSoftware Co-Engineering in t...

CONDUCTIV.AI PROCESS CONTROL

The Conductiv.ai Process Control makes it incredibly easy to model complex processes and create Self-Perfecti...

39 Tools

Figure 3: The Tools catalogue.

3. Servitization of certifying model checking

3.1. Context and motivations

The application of formal methods in the certification of high-assurance systems demands the qualification of the verification tools to ensure a sufficient level of confidence in their results (see for example the DO-333 standard in the avionic domain). Verification tools such as model checkers are, however, typically quite complex pieces of software, implementing sophisticated algorithms and heuristics in order to ensure maximum efficiency and scalability, which makes their qualification extremely challenging. An alternative, potentially simpler approach is that of certifying the *results produced by the model checker,* rather than certifying the model-checking tool itself [1, 2]. In particular, an appealing idea is that of certifying model-checking results by means of the *generation of proof certificates* as a byproduct of the verification process. Such certificates can then be verified by independent checkers, which are usually simpler to certify than the proof-generating tools.

3.2. Overview of the approach

We focus on the problem of certifying the results of model checking *invariant properties*, that is, properties that must hold in all the reachable states of the input system. We represent transition systems in a *symbolic* way, as a pair of quantifier-free first-order formulas with background theories (such as those used in Satisfiability Modulo Theories – SMT [3]): a formula *I(X)* representing the initial states of the system (where *X* is the set of state variables), and a formula *T(X, X')* representing the transition relation of the system (where the variables *X'* denote the state variables after performing one transition). Given a system M = <I(X), T(X, X')> and an invariant property P(X) - which is simply a formula over the state variables X, the invariant model checking problem (denoted with -
M |= P) can be reduced to finding an *inductive invariant* [4], that is a formula Inv(X) satisfying the following conditions: (1) it is implied by the initial-states formula I(X): |= I(X) -> Inv(X); (2) it is preserved by the transition relation: |= (Inv(X) & T(X, X')) -> Inv(X'); and (3) it implies the property: |= Inv(X) -> P(X). Inductive invariants can be generated automatically by state-of-the-art SAT- and SMT-based model checking algorithms, without any additional overhead. We can use an inductive invariant to define a *proof certificate* for M |= P as the SMT formula consisting of the conjunction of the three conditions above: (I(X) -> Inv(X)) & (Inv(X) & T(X, X') -> Inv(X')) & (Inv(X) -> P(X)). Checking the proof certificate amounts to proving the validity of the formula, a task which can be performed with any off-the-shelf SMT solver.

3.3. SMT and VMT formats

In order to maximize interoperability, our implementation relies on standard formats for the representation of both model checking problems and proof certificates. In particular, we leverage the SMT-LIB format (http://smt-lib.org) [5], the standard input language of SMT solvers, for the representation of the formulas encoding the input symbolic transition system and the validity check for the proof certificate. For the representation of the transition system, we rely on the VMT format (http://vmt-lib.org), an extension of the SMT-LIB format specifically designed to represent symbolic transition systems and their properties. VMT exploits the capability offered by the SMT-LIB language of attaching *annotations* to terms and formulas in order to specify the components of the transition system and the properties to verify. As such, a VMT file is a valid SMT-LIB file, with no special syntax needed. Compared to other SMT-LIB extensions with similar goals, VMT has therefore the benefit that it can be handled by the many available tools for manipulating SMT-LIB files that are already available for several programming languages.

3.4. OSLC-based interaction

The FBK Tools (namely OCRA, nuXmv, xSAP, ic3ia and mathsat5) expose an API that allows the integration with any kind of OSCL client. Such API extends the OSLC Automation specification that is defined by the Open Services for Lifecycle Collaboration (OSLC) Core Specification. The full specification is available at:

http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.1/

OSLC is a set of specifications that simplify the tool integration; it is based on Web standards (W3C Linked Data and RESTful Web Services). The OSLC specifications are organized in OSLC Domains, that are specifications leveraging the OASIS OSLC lifecycle integration Core Specification and enabling interoperation through the specification of standard domain vocabularies, constraints, and services.

The FBK Tools OSLC API is based on the Automation domain that defines the resources and operations supported by an Open Services for Lifecycle Collaboration (OSLC) **Automation Service Provider**. Automation resources define automation plans, automation requests and automation results of the software development, test, and deployment lifecycle.

The relationships between Automation Resources is depicted in Figure 4:

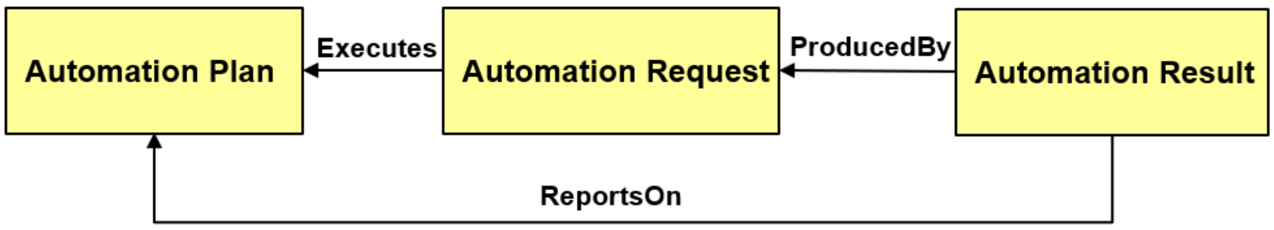


Figure 4: Relationships between Automation Resources.

For the FBK tools, a specific OSLC Service Provider based on the OSLC Automation domain has been realized. Figure 5 shows the interactions between the OSCL Client (CHESS in this example) and the Service Provider:

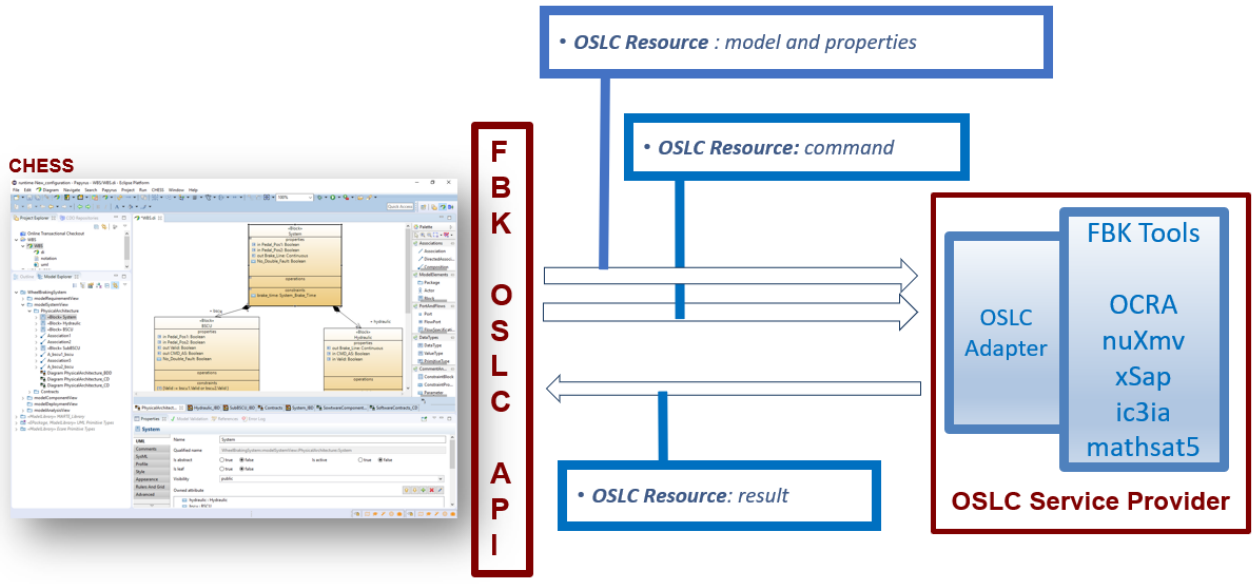


Figure 5: Interactions between the OSCL Client.

The functionality for the certification of Model Checking, that is proposed in this document, is composed of three steps that are performed in sequence:

1. Generation of VMT model from the SMV Model
2. Generation of SMT Proof Obligations from the VMT Model and Property
3. Checking of SMT Proof Obligations

Each step involves a different FBK Tool as shown in Figure 6:

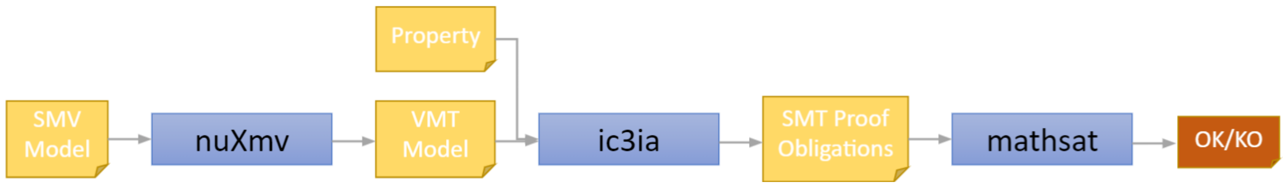
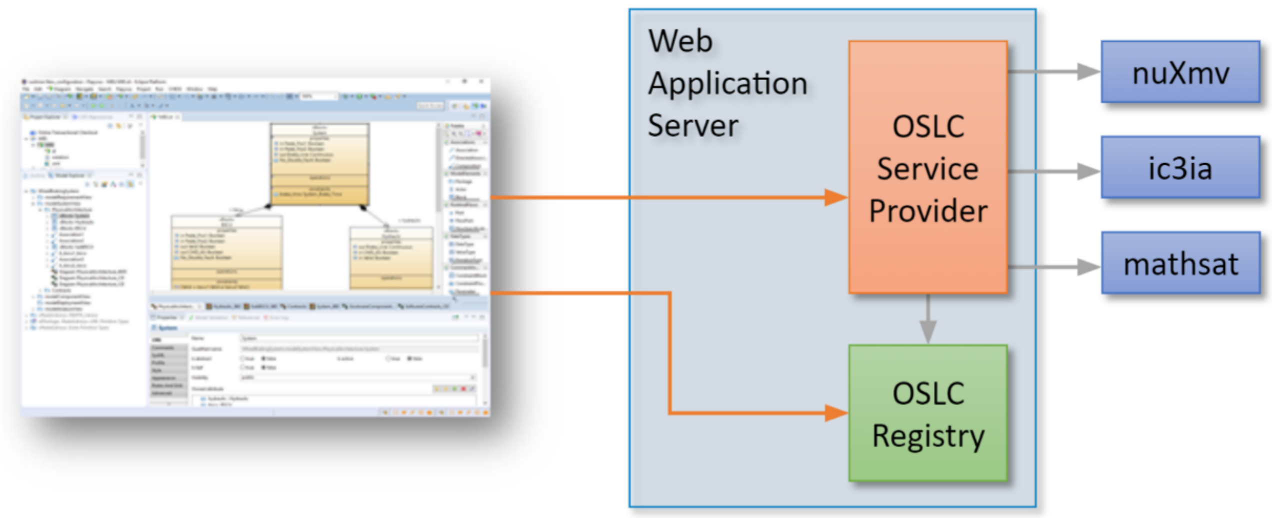


Figure 6: Steps involve in different FBK Tool.

The simplest integration and implementation of the tool chain through OSLC is realized by a web application server that runs the OSCL Registry (that lists the available services) and the OSLC Service Provider that interacts with the analysis tools.

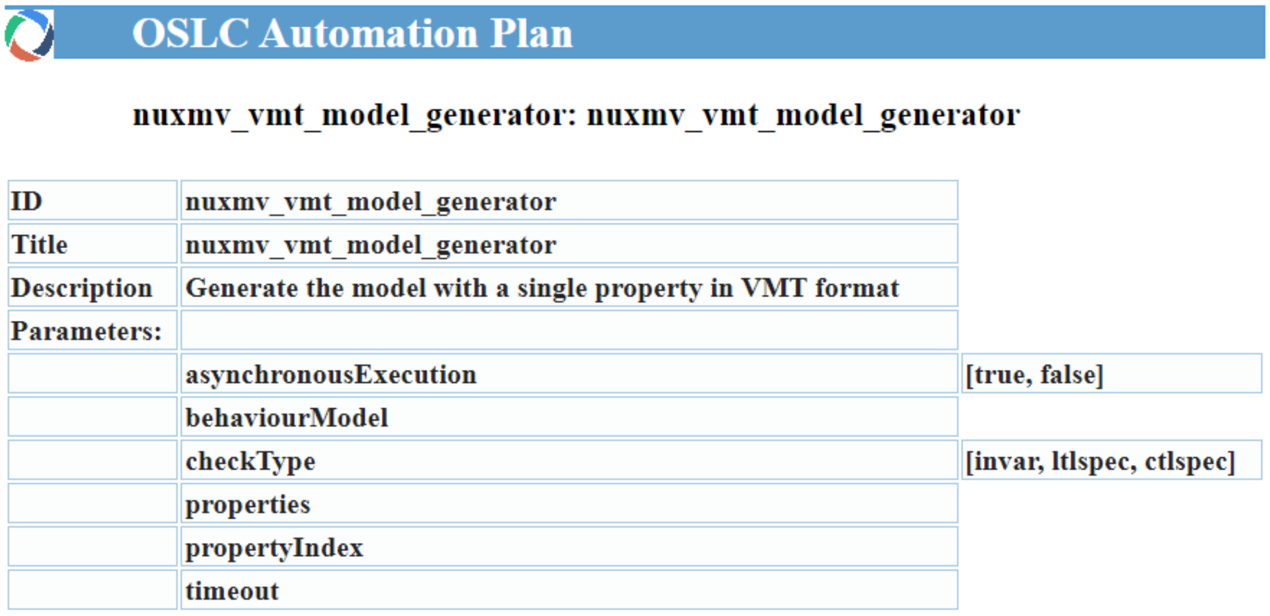


The Plans

The FBK tool functionalities are made available by means of Automation Plans; the title attribute of the Automation Plan selects the tool functionality. For the certification of Model Checking, the available plans are the following:

FBK Tool Functionality	**Automation Plan Title**
Generation of VMT Model	`nuxmv_vmt_model_generator`
Generation of SMT Proof Obligations	`ic3ia_proof_generation`
Check of SMT Proof Obligations	`check_satisfiability`

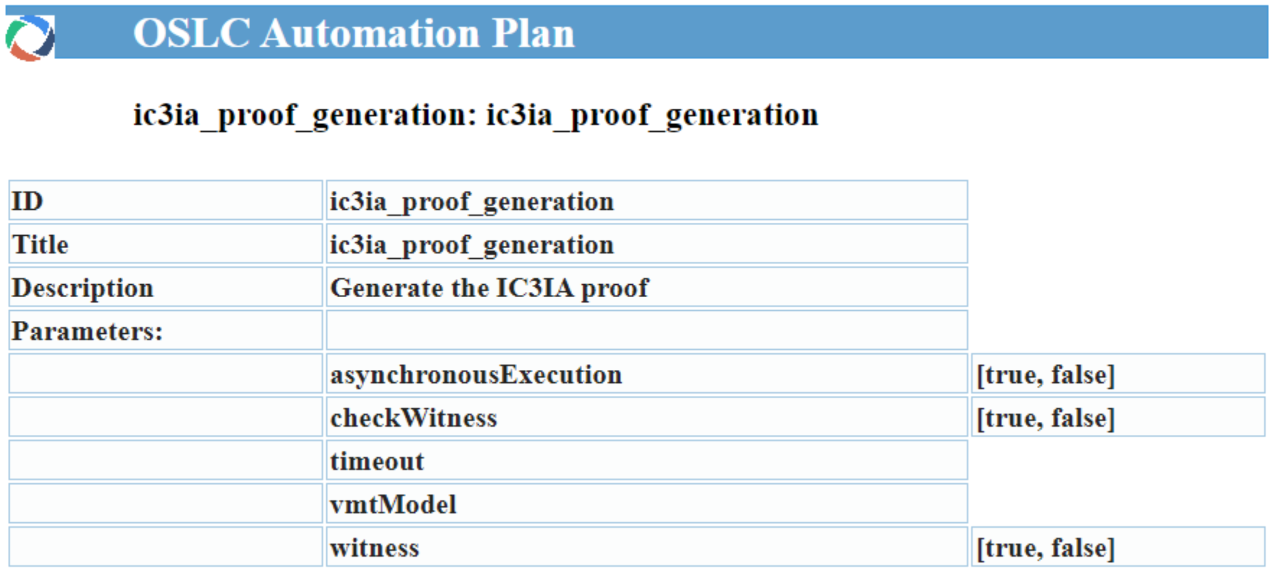
Each tool function admits some input parameters to be executed; they are listed following:



OSLC Automation Plan

nuxmv_vmt_model_generator: nuxmv_vmt_model_generator

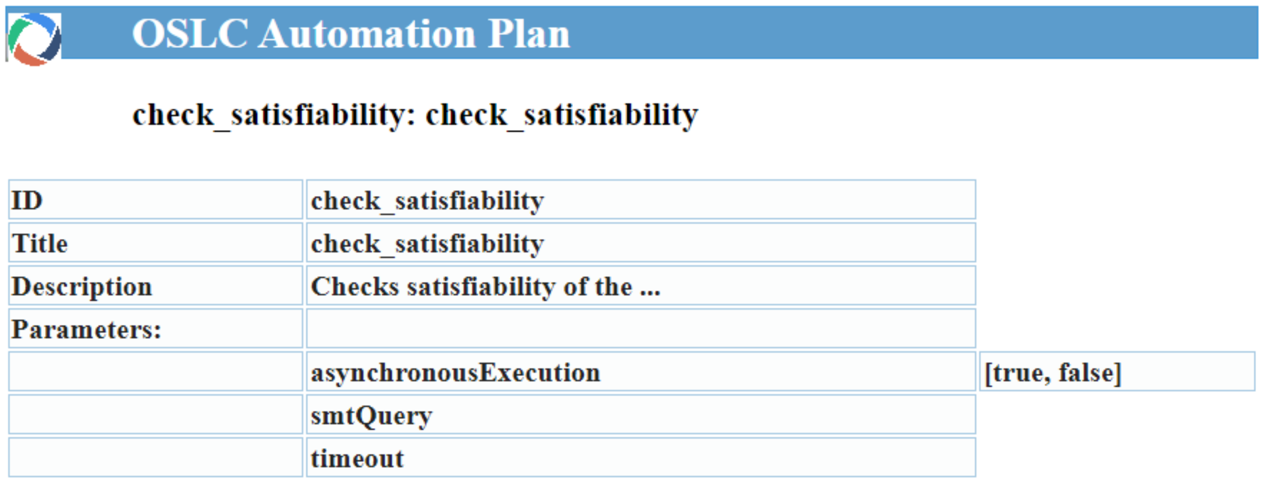
ID	nuxmv_vmt_model_generator	
Title	nuxmv_vmt_model_generator	
Description	Generate the model with a single property in VMT format	
Parameters:		
	asynchronousExecution	[true, false]
	behaviourModel	
	checkType	[invar, ltlspec, ctlspec]
	properties	
	propertyIndex	
	timeout	



OSLC Automation Plan

ic3ia_proof_generation: ic3ia_proof_generation

ID	ic3ia_proof_generation	
Title	ic3ia_proof_generation	
Description	Generate the IC3IA proof	
Parameters:		
	asynchronousExecution	[true, false]
	checkWitness	[true, false]
	timeout	
	vmtModel	
	witness	[true, false]



OSLC Automation Plan

check_satisfiability: check_satisfiability

ID	check_satisfiability	
Title	check_satisfiability	
Description	Checks satisfiability of the ...	
Parameters:		
	asynchronousExecution	[true, false]
	smtQuery	
	timeout	

Plan Execution Steps

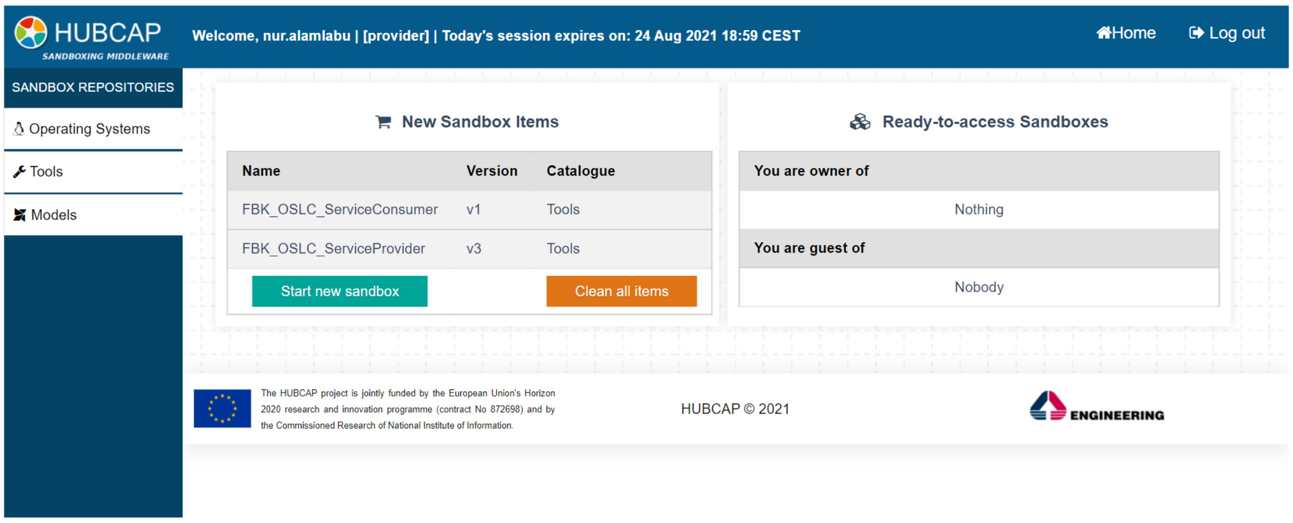
The steps to follow for executing a remote function of the FBK tools via OSLC API are:

1. Find in the catalogue the URI of Service Provider
2. Ask to the Service Provider the Automation Plan that maps the desired tool function.
3. Create an Automation Request that links such Plan, set the input parameters (see the Plan for their definitions) and submit the Request to the service provider.
4. Once the request has been submitted, its status can be “in progress” or “completed” depending on if the request is synchronous or asynchronous.
5. If the request is synchronous, when the Request is completed, ask the Service Provider for the of the Automation Result that is linked to the Request. (See the Result property <`oslc_auto:producedByAutomationRequest`>)
6. If the request is asynchronous, poll the Service Provider by asking the status of the Request (see property <`oslc_auto:state`>) and, if completed, ask the Service Provider for the Automation Result that is linked to the Request.

3.5. Experiment within a sandbox

The following section evaluates the experiment of the OSLC based interaction within a sandbox, with snapshots of the sequence. We need two virtual machines (VM) to experiment with the OSLC base interaction in a sandbox.

As the first step, we select two tools named "FBK_OSLC_ServiceConsumer" for the SDE that consume the service and "FBK_OSLC_ServiceProvider" for the service provider from the Tools repositories from the left sidebar on the home page.


HUBCAP SANDBOXING MIDDLEWARE

Welcome, nur.alamlabu | [provider] | Today's session expires on: 24 Aug 2021 18:59 CEST

Home Log out

SANDBOX REPOSITORIES
- Operating Systems
- Tools
- Models

New Sandbox Items

Name	Version	Catalogue
FBK_OSLC_ServiceConsumer	v1	Tools
FBK_OSLC_ServiceProvider	v3	Tools

Start new sandbox Clean all items

Ready-to-access Sandboxes

You are owner of
Nothing
You are guest of
Nobody

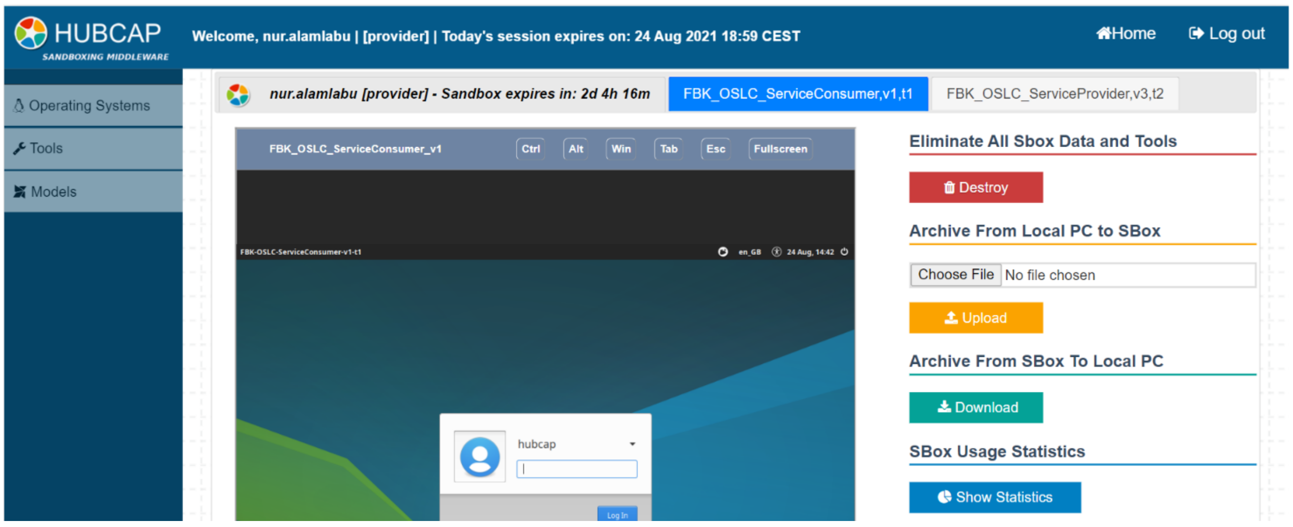
The HUBCAP project is jointly funded by the European Union's Horizon 2020 research and innovation programme (contract No 872698) and by the Commissioned Research of National Institute of Information.

HUBCAP © 2021

ENGINEERING

Figure 7: A cart (New Sandbox Items section) on the home page with two Tools.

Once both tools are selected, we can launch the new sandbox by clicking on the "Start new sandbox" button, available in the "New Sandbox Items" section of the home page (Figure 7). The Sandboxing Middleware will start to instantiate the desired Tools within a new Sandbox.


HUBCAP SANDBOXING MIDDLEWARE

Welcome, nur.alamlabu | [provider] | Today's session expires on: 24 Aug 2021 18:59 CEST

Home Log out

- Operating Systems
- Tools
- Models

nur.alamlabu [provider] - Sandbox expires in: 2d 4h 16m

FBK_OSLC_ServiceConsumer,v1,t1 FBK_OSLC_ServiceProvider,v3,t2

FBK_OSLC_ServiceConsumer_v1 Ctrl Alt Win Tab Esc Fullscreen

FBK-OSLC-ServiceConsumer-v1-t1 en_GB 24 Aug, 14:42

hubcap

Log In

Eliminate All Sbox Data and Tools

Destroy

Archive From Local PC to SBox

Choose File No file chosen

Upload

Archive From SBox To Local PC

Download

SBox Usage Statistics

Show Statistics

Figure 8: Two instances of two tools running in the Sandbox.

Once the instantiation process is complete, the Sandbox Viewer will be prompted (Figure 8). The Tool selector in the top area of Sandbox Viewer shows two separate Tools (Two different VMs called "FBK_OSLC_ServiceConsumer" and "FBK_OSLC_ServiceProvider").

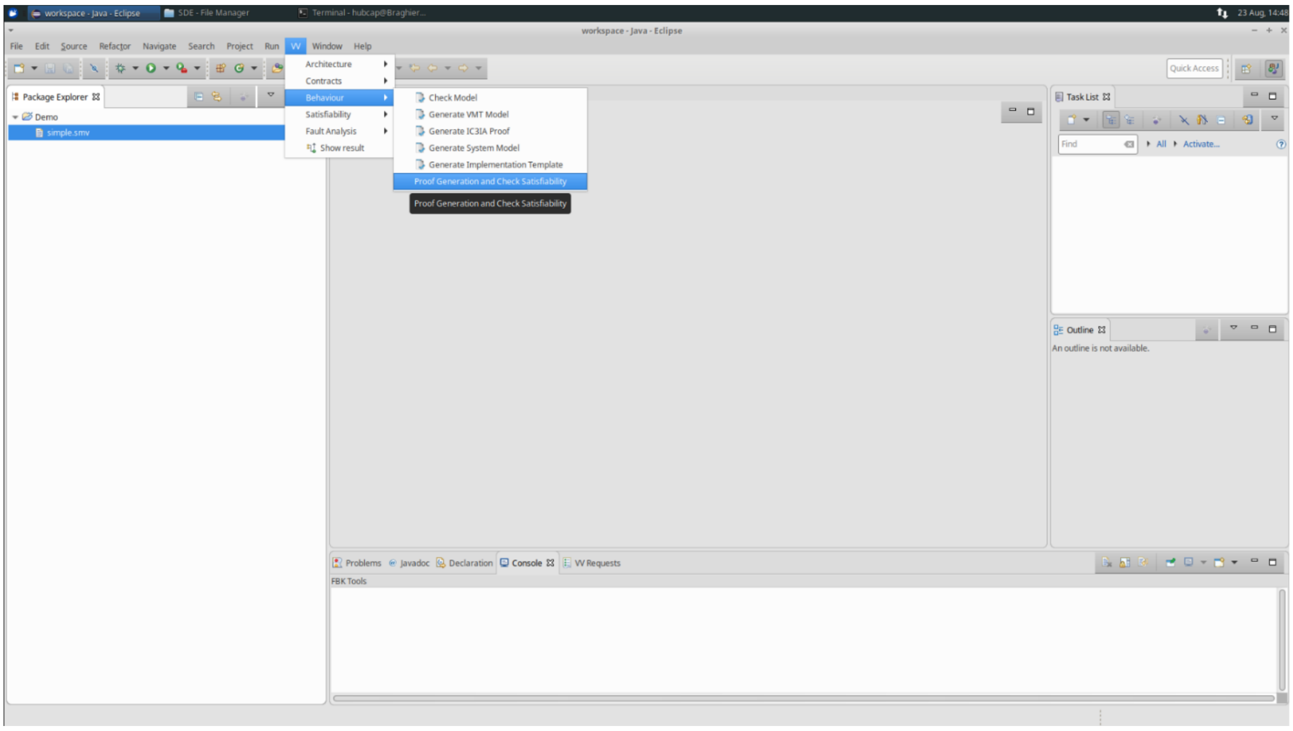
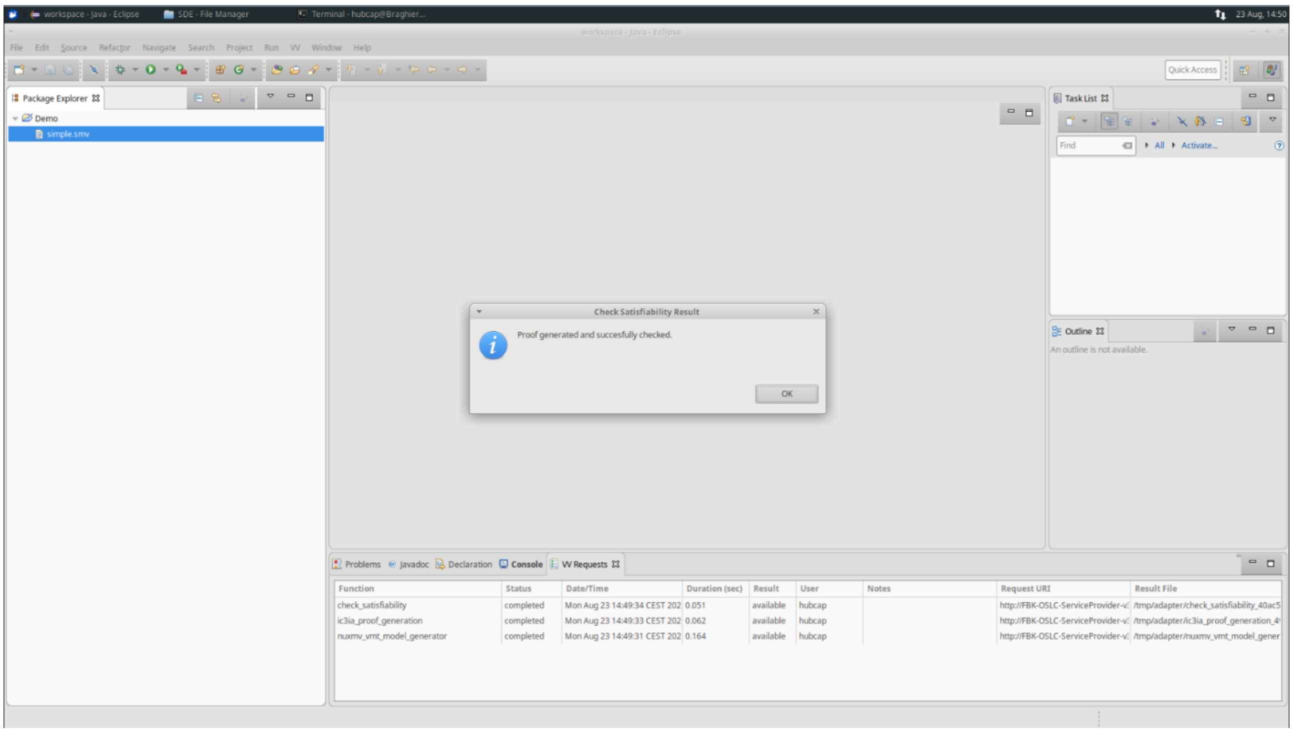


Figure 9: Eclipse SDE run in the "FBK_OSLC_ServiceConsumer" tool.

Once the sandbox is ready, we run eclipse in the "FBK_OSLC_ServiceConsumer" VM then select the "simple.smv" file in the Demo project and after that select the menu VV->Behaviour->Proof Generation and Check Satisfiability (Figure 9).

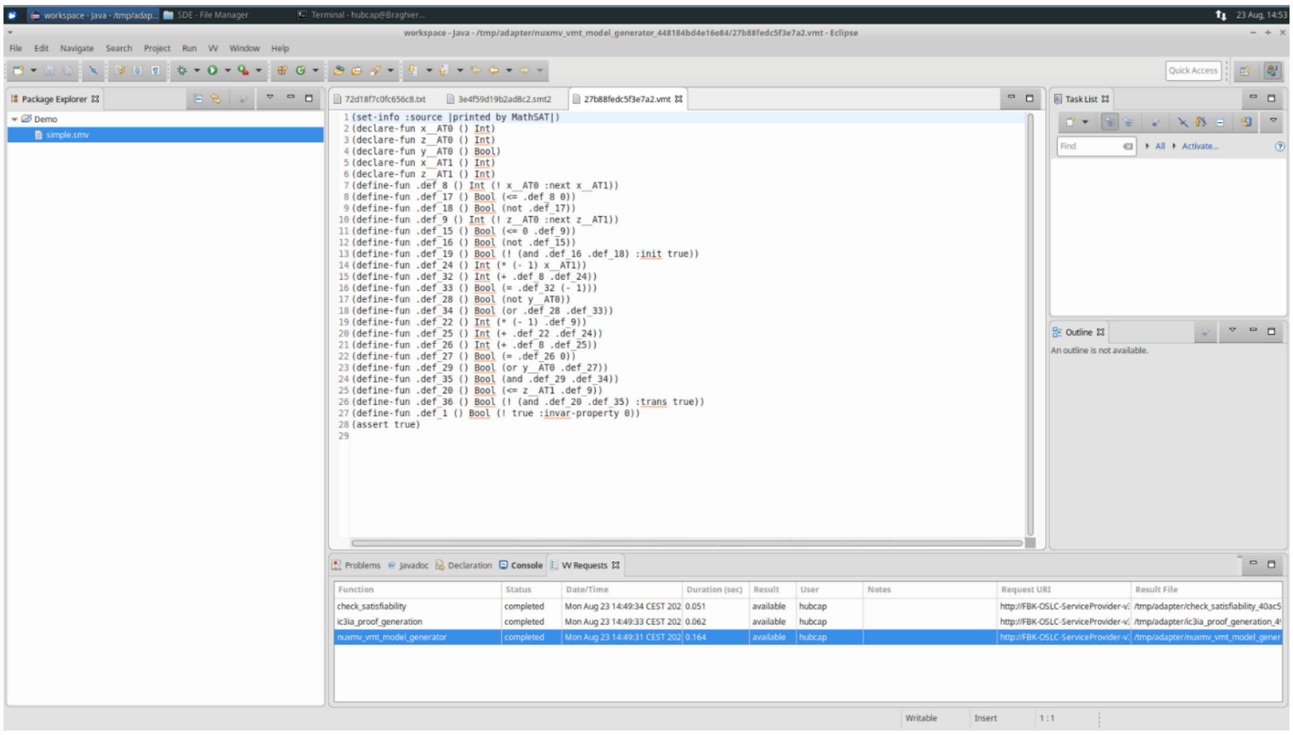
One message will arrive in a dialog box when the proof generation and check satisfiability run successfully (Figure 10).


Check Satisfiability Result
Proof generated and succesfully checked.
OK
Package Explorer
Demo
simple.smv
Task List
Outline
An outline is not available.
Problems
Javadoc
Declaration
Console
VV Requests

Function	Status	Date/Time	Duration (sec)	Result	User	Notes	Request URI	Result File
check_satisfiability	completed	Mon Aug 23 14:49:34 CEST 202	0.051	available	hubcap		http://FBK-OSLC-ServiceProvider-v:	/tmp/adapter/check_satisfiability_40ac5
ic3ia_proof_generation	completed	Mon Aug 23 14:49:33 CEST 202	0.062	available	hubcap		http://FBK-OSLC-ServiceProvider-v:	/tmp/adapter/ic3ia_proof_generation_4
nuxmv_vmt_model_generator	completed	Mon Aug 23 14:49:31 CEST 202	0.164	available	hubcap		http://FBK-OSLC-ServiceProvider-v:	/tmp/adapter/nuxmv_vmt_model_gener

Figure 10: Message for the successful run.

The VV Request view will appear with three OSLC invocations to FBK tools (namely OCRA, nuXmv, xSAP, ic3ia and mathsat5) that are running in the second VM called "FBK_OSLC_ServiceProvider" (Figure 11).


workspace - Java - /tmp/adapter/nuxmv_vmt_model_generator_448184bd4e16e84/27b88fedc5f3e7a2.vmt - Eclipse
72d18f7c0fc656c8.txt
3e4f59d19b2ad8c2.smt2
27b88fedc5f3e7a2.vmt
Package Explorer
Demo
simple.smv
Task List
Outline
An outline is not available.
Writable
Insert
1 : 1

```
(set-info :source |printed by MathSAT|)
(declare-fun x__AT0 () Int)
(declare-fun z__AT0 () Int)
(declare-fun y__AT0 () Bool)
(declare-fun x__AT1 () Int)
(declare-fun z__AT1 () Int)
(define-fun .def_8 () Int (! x__AT0 :next x__AT1))
(define-fun .def_17 () Bool (<= .def_8 0))
(define-fun .def_18 () Bool (not .def_17))
(define-fun .def_9 () Int (! z__AT0 :next z__AT1))
(define-fun .def_15 () Bool (<= 0 .def_9))
(define-fun .def_16 () Bool (not .def_15))
(define-fun .def_19 () Bool (! (and .def_16 .def_18) :init true))
(define-fun .def_24 () Int (* (- 1) x__AT1))
(define-fun .def_32 () Int (+ .def_8 .def_24))
(define-fun .def_33 () Bool (= .def_32 (- 1)))
(define-fun .def_28 () Bool (not y__AT0))
(define-fun .def_34 () Bool (or .def_28 .def_33))
(define-fun .def_22 () Int (* (- 1) .def_9))
(define-fun .def_25 () Int (+ .def_22 .def_24))
(define-fun .def_26 () Int (+ .def_8 .def_25))
(define-fun .def_27 () Bool (= .def_26 0))
(define-fun .def_29 () Bool (or y__AT0 .def_27))
(define-fun .def_35 () Bool (and .def_29 .def_34))
(define-fun .def_20 () Bool (<= z__AT1 .def_9))
(define-fun .def_36 () Bool (! (and .def_20 .def_35) :trans true))
(define-fun .def_1 () Bool (! true :invar-property 0))
(assert true)
```

Function	Status	Date/Time	Duration (sec)	Result	User	Notes	Request URI	Result File
check_satisfiability	completed	Mon Aug 23 14:49:34 CEST 202	0.051	available	hubcap		http://FBK-OSLC-ServiceProvider-v:	/tmp/adapter/check_satisfiability_40ac5
ic3ia_proof_generation	completed	Mon Aug 23 14:49:33 CEST 202	0.062	available	hubcap		http://FBK-OSLC-ServiceProvider-v:	/tmp/adapter/ic3ia_proof_generation_4
nuxmv_vmt_model_generator	completed	Mon Aug 23 14:49:31 CEST 202	0.164	available	hubcap		http://FBK-OSLC-ServiceProvider-v:	/tmp/adapter/nuxmv_vmt_model_gener

Figure 11: The result from VV Requests.

4. Diversification of analysis

A way to ensure that your system is modeled properly is to implement it in two different variants (diversification) either using the same or different paradigms. When using the same paradigm, using the same formalism and tool, it is often that case that two independently working teams are tasked with the same analysis goals. A redundant team is also used, but not necessarily when different paradigms are used to perform modeling and analysis using different methods or tools, thus leading to alternative variants of the same result. By analyzing the results obtained via running the variants one can reason about the correctness of the implemented solution. If the different variants lead to different analysis results, one of the following situations has arisen:

- the requirements were interpreted differently for the two systems (thus they might be ambiguous)
- one of the implementations is wrong and one is correct
- both implementations are wrong

If the results are equal or sufficiently similar, then one can say that:

- the various implementations of the system are most likely correct
- the various implementations of the system are all wrong and both are making the same mistake (which might come from an ambiguity in the requirements)

To illustrate a diversification of analysis exercise using different teams and paradigms we have implemented the same system, a water tank controller using:

- Modelica in OpenModelica
- The Vienna Development Method using the Overture tool
- SIDOPS+ in 20sim

4.1. Example - Single Water Tank

The single-tank water is a simple example that comprises a single water tank which is controlled by a cyber controller. When the water level of the tank reaches a particular level (defined in the controller) the controller sends a command to the system to open an exit valve, thus lowering the water level in the tank. A diagram of the example is shown in Figure 12.

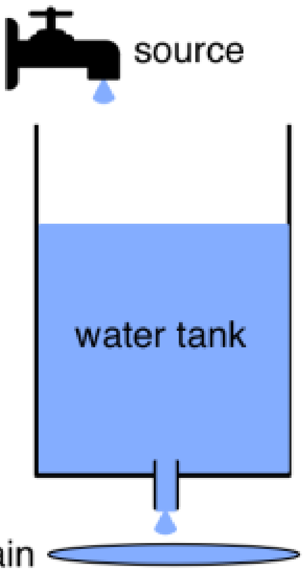


Figure 12. Single Water Tank example

We have implemented this system using several languages and modeling paradigms and compared the simulation results.

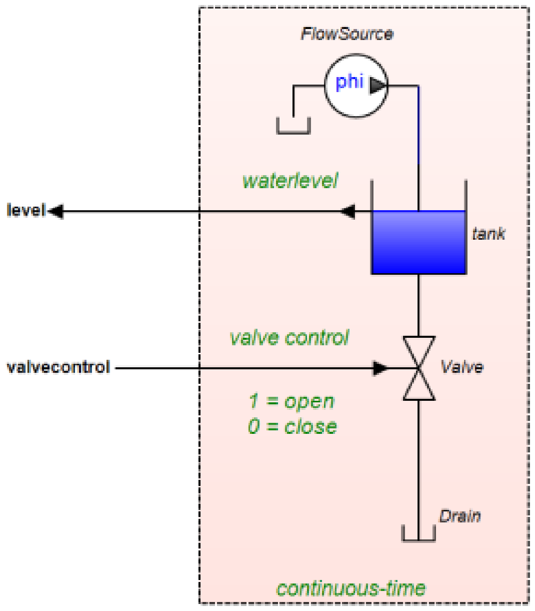
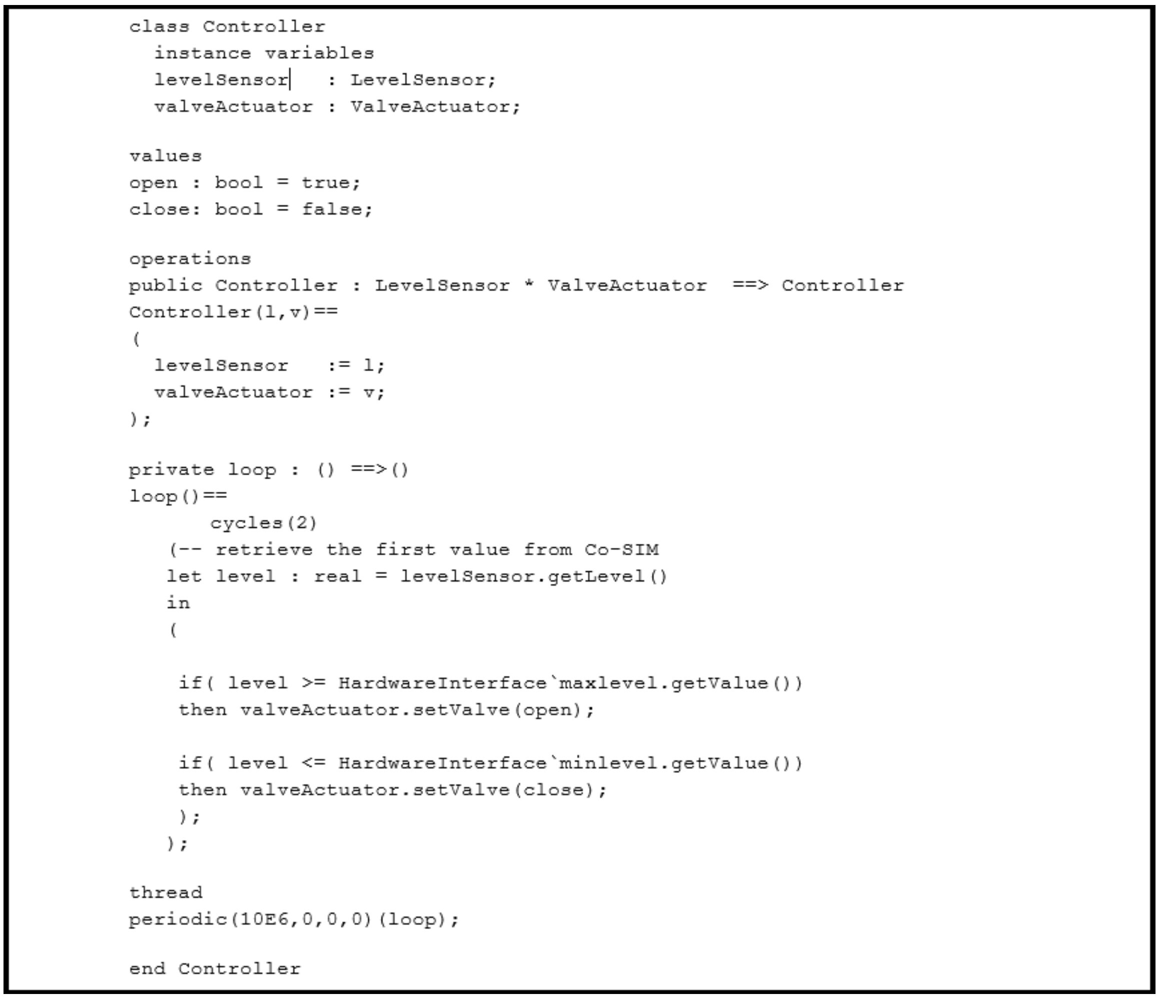


Figure 13. Single Water Tank example model in 20sim

The water tank model from Figure 13 was implemented in 20sim in a previous project and the source code can be found at: https://gitlab.au.dk/into-cps/github-into-cps/case-study_single_watertank/-/blob/master/Models/Watertank.emx

The controller used to control the tank was implemented using VDMRT in Overture and its specification is given below Figure 14:



```
class Controller
  instance variables
  levelSensor   : LevelSensor;
  valveActuator : ValveActuator;

values
open : bool = true;
close: bool = false;

operations
public Controller : LevelSensor * ValveActuator  ==> Controller
Controller(l,v)==
(
  levelSensor   := l;
  valveActuator := v;
);

private loop : () ==>()
loop()==
      cycles(2)
   (-- retrieve the first value from Co-SIM
   let level : real = levelSensor.getLevel()
   in
   (

    if( level >= HardwareInterface`maxlevel.getValue())
    then valveActuator.setValve(open);

    if( level <= HardwareInterface`minlevel.getValue())
    then valveActuator.setValve(close);
    );
   );

thread
periodic(10E6,0,0,0)(loop);

end Controller
```

Figure 14: Controller code for VDMRT in Overture.

Both the Controller model and the WaterTank model were compiled into FMUs and co-simulated using the INTO-CPS COE. The results are presented in Figure 15.

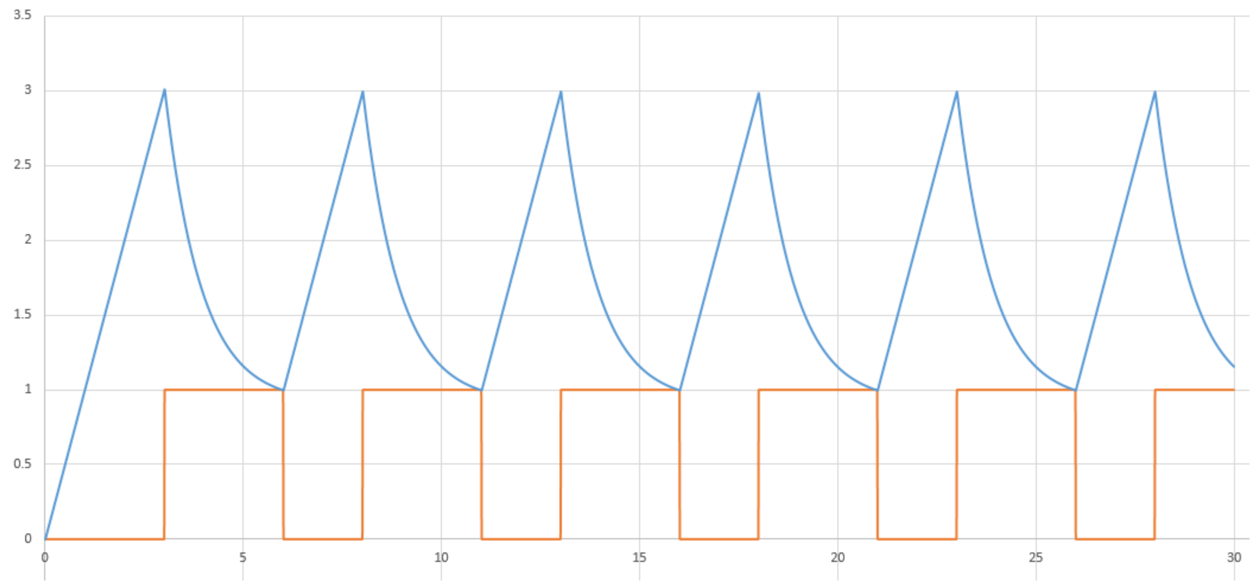


Figure 15. Co-simulation using FMUs from Overture for control and 20sim for the tank

The same water tank system was modeled using Modelica for both the controller and the tank and then a test system was simulated using OpenModelica. The Modelica code is given further below. The visual representation of the system in Modelica is shown in Figure 16.

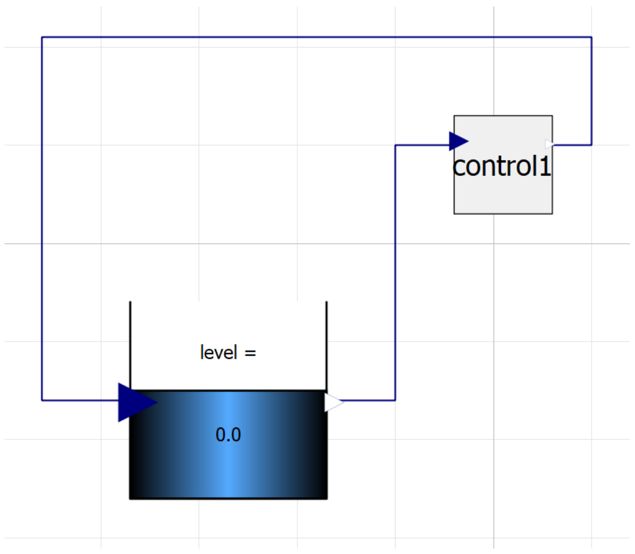


Figure 16. Single Water Tank example model in Modelica

The simulation of the system gave the results from Figure 17.

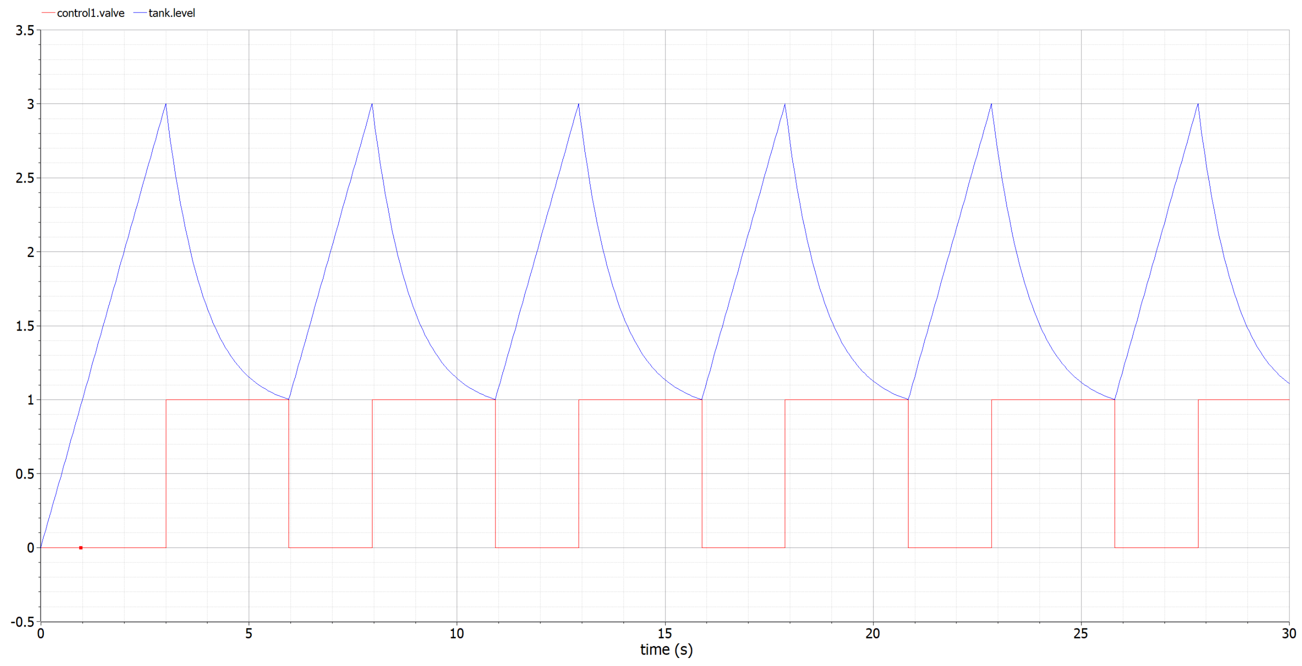


Figure 17. Simulation of the Single Water Tank example model in OpenModelica

The Modelica package containing the “SingleWaterTank” model, the “Control” model and the “TestSingleWaterTank” model is listed in the following:

```
package WaterTank
  model SingleWaterTank
    import Modelica.Constants.pi;
    import SI = Modelica.SIunits;
    // Tank
    parameter SI.Area area = 1.0 "Area of tank";
    parameter Real gravity = 9.81;
    parameter Real level_start = 0.0;
    parameter Real liquid_density = 1;
    parameter Real volume_initial = 0;
    Real volume(start = volume_initial);
    Real tank_p_f;
    Real tank_p_e;
    Real tank_p_phi;
    // Flow source
    parameter Real fs_phi = 1.0;
    Real fs_p_phi = fs_phi;
    Real fs_inflow;
    Real fs_p_p;
    // Valve
    parameter Real valve_outflow_int_initial = 0;
    Real valve_outflow_int(start = valve_outflow_int_initial);
    Real valve_puddle;
    Real valve_powerOut_e;
    Real valve_powerOut_f;
    Real valve_powerIn_phi;
    Real valve_powerIn_p;
    Real valve_outflow;
    // Drain
    Real drain_p_e;
    Real drain_p_f;
    parameter Real drain_r = 9;
    Modelica.Blocks.Interfaces.RealInput valvecontrol;
    Modelica.Blocks.Interfaces.RealOutput level;
  equation
    fs_inflow = if fs_p_phi < 0 then 0 else 0.1 * abs(fs_p_phi);
    tank_p_e = gravity * volume * liquid_density;
    level = volume / area;
    valve_puddle = 8 * sqrt(valve_outflow_int);
    valve_powerOut_e = if valvecontrol <> 0.0 then tank_p_e else 0.0;
    drain_p_f = valve_powerOut_e / drain_r;
    valve_powerIn_phi = if valvecontrol <> 0.0 then drain_p_f else 0.0;
    valve_outflow = if drain_p_f < 0.0 then 0 else 0.1 * abs(drain_p_f);
    tank_p_f = fs_p_phi - valve_powerIn_phi;
    der(volume) = tank_p_f;
    der(valve_outflow_int) = valve_outflow;
    tank_p_phi = fs_p_phi;
    fs_p_p = tank_p_e;
    valve_powerIn_p = tank_p_e;
    drain_p_e = valve_powerOut_e;
    valve_powerOut_f = drain_p_f;
  end SingleWaterTank;

  model Control
    parameter Real minlevel = 1;
    parameter Real maxlevel = 3;

    Modelica.Blocks.Interfaces.RealInput level;
    Modelica.Blocks.Interfaces.RealOutput valve(start = 0);
  equation
    valve = if level >= maxlevel then 1.0
            elseif level < minlevel then 0.0
            else pre(valve);
  end Control;

  model TestSingleWaterTank
    Control control1;
    SingleWaterTank tank;
  equation
    connect(control1.valve, tank.valvecontrol);
    connect(tank.level, control1.level);
  annotation(
      uses(Modelica(version = "3.2.3")),
      experiment(StartTime = 0, StopTime = 30, Tolerance = 1e-06, Interval = 0.06));
  end TestSingleWaterTank;
end WaterTank;
```

Discussion

When comparing the system's response in Figure 15 and Figure 17, one observes the same results (apart from minor negligible differences due to arithmetic precision). When the tank reaches the maximum desired water level the controller opens the valve and when the water level reaches the minimum desired level the valve is closed. The two plots can be made to coincide with any precision, but we have opted to work with the current approximation given that our goal is to exemplify how to increase confidence in models by analyzing different variants.

Having similar results for the same system using diversification via variant implementations in different languages and paradigms can give confidence in the system specification and its implementations. Yet, it would be difficult to do an exhaustive comparison with all the available tools in the HUBCAP platform. We expect this example provides an alternative for future research directions.

5. TRL arguments

The metric of Technology Readiness Level (TRL) was defined as a relevant parameter to be included in HUBCAP with the intention to give its users and stakeholders a better understanding on the level of development of the tools that form part of the platform. TRL was developed by NASA in 1995 to assess the level of maturity of technologies intended to be utilized in space missions. As this perspective was centred on the assessment of technologies developed in the aerospace field, this chapter will centre on the process of development of a TRL adapted for HUBCAP tools.

5.1. Context and motivations

As previously mentioned, TRL is a metric/measurement defined to support the assessment of maturity level of a particular technology [6]. Initially, this metric was developed with the intention to assess technologies for the aerospace industry.

Based on the first document created by NASA to define the TRL metric, the model can be adapted to different environments considering the next 5 important characteristics [6]:

a. “Basic” Research in the technology or concept
b. Focused technology development addressing specific technologies or concepts for an application
c. Technological development of the application
d. System development (fabrication phase)
e. System “launch”

In the context of NASA, the model had initially 7 levels that some years later were extended to 9. The first level starts from a basic principle observed and reported and goes until the final level centred on the successful implementation of the system in operations. With time, TRL gained recognition in other fields of different industries. For this reason, it suffered different mutations to be adapted to the different areas. In the manufacturing industry, this metric gained high levels of attention, until being specifically addressed by the European Commission and was defined in the context of EU Horizon programs slightly different from the original definition of NASA as [7]:

TRL 1 – basic principles observed
TRL 2 – technology concept formulated
TRL 3 – experimental proof of concept
TRL 4 – technology validated in laboratory
TRL 5 – technology validated in relevant environment (industrially relevant environment in the case of key enabling technologies)

TRL 6 – technology demonstrated in relevant environment (industrially relevant environment in the case of key enabling technologies)
TRL 7 – system prototype demonstration in operational environment
TRL 8 – system complete and qualified
TRL 9 – actual system proven in operational environment (competitive manufacturing in the case of key enabling technologies or in space)

In the EU Horizon programs context, the TRLs were defined with the intention to narrow down the topics of the H2020. Additionally, the definition of this metric, gives an idea on the next steps that the project must follow and an initial iteration on the distance of the technology from the market.

In the context of HUBCAP, the definition of the TRLs became vital due to the fact that it can support stakeholders to define the current development step of the tool in discussion, identify its possible benefits and limitations and give a broad idea on the further steps that it will have.

5.2. Guidelines for HUBCAP providers

Starting from the TRL definition made by the EU Commission and considering in addition the definition of each level made in the original white paper by NASA, each TRL was adapted to the tool context of HUBCAP. The main goal of the activity was not to modify in high level the definitions initially made by NASA. This with the intention to avoid loses in its concreteness and sophisticated perspective that it has from its initial application in aerospace. This due to the fact that the modification of the model can bring confusion if it is done to a high extent from its initial source [2].

Initially, the information regarding each one of the levels was first divided in columns to make it more readable and clearer to understand for the tool providers. To this end, 6 columns were created (later in the process expanded to 8):

- **Phase:** it defines a cluster where one or more levels are gathered and contain a common global objective
- **TRL:** number of the technology readiness level
- **Development stages:** name of the TRL
- **Development action taken:** it describes the action that should had been taken by the tool provider in order to ensure that they already reached each level
- **Cost of development:** an indicator of the usual costs that the level requires. This was taken specifically from the original white paper, but it can vary depending on the technology evaluated.
- **What to consider and examples:** this column describes facts to be considered while reflecting if the TRL analyzed is the correct one. Additionally, an example is given in each one of the levels
- **What question should the tool provider answer:** a question for the tool provider is defined to support their assessment on the definition of the correct TRL.
- **Output:** an output is defined for each one of the TRL. These outputs are recommendations as the type of reports and results can vary from provider to provider. For this reason, these recommendations can be presented in different formats or can be previous documents that maintain the same level of assurance of the TRL.

The process of design of the HUBCAP TRL is shown in the Figure 18:

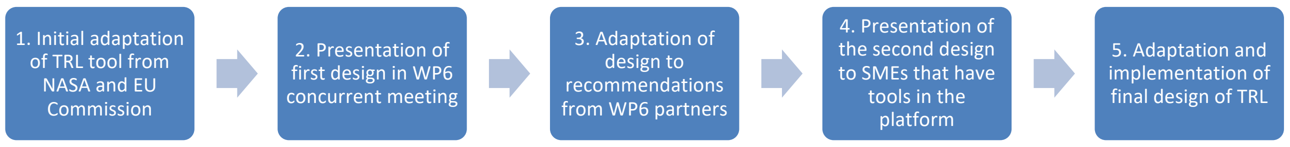

1. Initial adaptation of TRL tool from NASA and EU Commission
2. Presentation of first design in WP6 concurrent meeting
3. Adaptation of design to recommendations from WP6 partners
4. Presentation of the second design to SMEs that have tools in the platform
5. Adaptation and implementation of final design of TRL

Figure 18: TRL adaptation process for HUBCAP context.

1. The initial adaptation of TRL was done starting from the NASA and EU commission versions.
2. It was necessary to perform additional activities in order to verify the suitability of the TRL with the tools in HUBCAP. For this reason, some presentations of the tool were planned and executed. The first presentation was made in the usual status meetings of WP6.
3. The feedback gathered from the presentation in the WP6 meeting were:
   a. Inclusion of output for each level of the TRL. The output was defined generally as it can have high variations from one to other company.
   b. Corrections and clarifications of the definitions of each level. More specifically, the column "Development actions taken" were expanded to clarify them.
4. The second presentation was done as an additional activity where the tool was explained and commented with the SMEs that produce the tools. As these are the users of the tool, some additional corrections in terms of description and levels specifications were gathered.
5. The new feedback was implemented in the tool definition and then in the platform.

The resulting TRLs adapted to the HUBCAP context can be seen in the next tables.

Table 1. TRL definition in HUBCAP context - Part A

Phase	**Phase 1**		
TRL Number	**1**	**2**	**3**
Development stages	**Basic principles observed and reported**	**Tool concept and/or tool formulation**	**Analytical and experimental critical function and/or characteristic proof of concept**
Development action taken	Scientific research translated into applied research and development. Most relevant topics related with the tool end goal were investigated and applied to the tool methodology. This includes physical principles or basic topics over which the intended tool will be built on.	The tool is discussed and defined around the basic principles defined in the previous level. The previous topics converge in the definition of the tool objective, being merely speculative: without experimental proof or detailed analysis that supports the conjecture	Active R&D starts. Starting from the basic principles investigated and the definition of the tool, an appropriate context of the tools is defined, and studies are performed to validate that the analytical predictions are correct. This is translated into a "Proof of concept" of the tool concept.
Cost of development	Very low "unique" cost	Very low "unique" cost	Very low "unique" cost
What to consider in this stage and examples	The basic principles of the tool are researched. Ex: The analysis of the different types of the battery control methodologies	Practical applications of the basic principles are invented or identified. Ex: the potential application of a certain control methodology is explored, like its utilization for electric trucks	A proof of the concept over which the tool is build had been developed. Ex: If the tool is based on the implementation of a battery management system (BMS) for electric trucks, then this stage is achieved when it has been theoretically proven or modelled that the theoretical model behind the tool is feasible for the application. In the example: the theory behind the battery control has been proved.
What question should the tool provider answer?	[Is the tool basic principle already researched?]	[The applicability of the basic principles has been already explored?]	[Is the theory behind the tool and the application already theoretically proven for this type of applications?]

Output	Documents about basic physical principles or basic topics over which the intended tool will be built on. Further information about how scientific research (with related references) has been translated into applied research and development.	Document reporting the tool objective connected with the physical principles and basic topics found in the previous stage.	Proof of concept of the tool. Document reporting the validation of the concept of the tool (feasibility analysis of tool/model) Document reporting requirements of potential real applications of the tool.

Table 2. TRL definition in HUBCAP context - Part B

Phase	**Phase 1**	**Phase 2**	
TRL Number	**4**	**5**	**6**
Development stages	**Tool concept validation in laboratory digital environment**	**Tool concept validation in relevant environment**	**Tool prototype demonstration in a relevant environment**
Development action taken	A validation in a digital environment must support the concept formulated in the previous step being consistent with the requirements of potential real applications	The basic tool elements must be integrated with reasonably realistic environments. The demonstration might represent an actual system application, or it might be like the planned application, but using the same technologies	A model or prototype system would be tested in a relevant environment. At this level, considering the NASA definition, if the only "relevant environment" is the space, then the model/prototype must be demonstrated in space. The demonstration might represent an actual system application, or it might only be like the planned application but using the same technologies. At this level, several-to-many new technologies might be integrated into the demonstration.
Cost of development	Low-to-moderate 'Unique' Cost	Moderate 'Unique' Cost	Technology and demonstration specific; a fraction
What to consider in this stage and examples	This validation can be done in a digital environment without considering all variables as in real application cases. Ex: Testing the tool with a virtual battery for a vehicle or truck	The validation in this stage must be done by implementing the tool in a similar environment than the application case. Ex: Testing the model for a truck battery control with a real truck or a similar vehicle	In this stage all the variables of the application are considered. Ex: In this stage, not only the model behind the tool must be tested, but also including additional variables such as movement, time, etc. which could affect the model of the tool.
What question should the tool provider answer?	[Is the Tool already tested in a digital environment proving that it is feasible to be applied?]	[Is the tool already tested and its results were taken to a similar environment to the real environment?]	[The previous test had been performed including external environment variables?]
Output	Report of the validation conducted in a digital environment	Tool prototype (Tool concept validated) Report of the validation of the tool concept in a relevant environment, showing conformity with the requirements previously defined.	Report of the demonstration of the tool prototype in a relevant environment considering additional variables of the application Validation of integrability with new technologies.

Table 3. TRL definition in HUBCAP context - Part C

Phase	**Phase 2**		**Phase 3**
TRL Number	**7**	**8**	**9**
Development stages	**Tool system prototype demonstration in an operation environment**	**Actual tool system completed with a test made in an environment similar to potential customer applications**	**Actual tool system proven to have successful application for customer needs**
Development action taken	This TRL is not always implemented. In this case, the prototype should be near or at the scale of the planned operational system and the demonstration must take place in the specific conditions of the real operational environment of the tool. The driving purposes for achieving this level of maturity are to assure system engineering and development management confidence (more than for purposes of technology R&D). Therefore, the demonstration must be of a prototype of that application. Not all technologies in all systems will go to this level. TRL 7 would normally only be performed in cases where the tool and/or subsystem application is mission critical and relatively high risk.	In this stage, the tool had been already tested in applications alike to the customer applications. In other words, the tool has been tested in a situation or application similar to the one that the potential customers would have.	All the tools that are being implemented by companies are in TRL 9. This stage is reached after **the end of the last "bug fixing" aspects of the tool development**.
Cost of development	Technology and demonstration specific, but a significant fraction of the cost of TRL 8	Typically, highest unique cost for a new technology	Mission Specific; less than cost of TRL 8

What to consider in this stage and examples	This stage is important in case that the tool is related with safety or critical risks for the customer company. Ex: If the battery control system includes the modelling of the support of security systems in the truck, the experiment must be performed including all the variables that can influence this.	The tool has to be tested in a real environment. Ex: Tested in a set of truck batteries.	The tool had been successfully implemented in real environments. Ex: The tool for battery control simulation had been tested in the application of an electric truck of a company with successful results in meeting customer needs.
What question should the tool provider answer?	[In case the tool has critical risk or security concerns, it has been tested in a real environment with the variables that affect these risks?]	[Is the tool already tested in a real environment as the ones that customers would use it on?]	[Is the tool already proven with customers and showed that it successfully meets their requirements?]
Output	Tool system prototype (Proven in risk environments) Document reporting the demonstration of the tool/system prototype in an operational environment granting that critical risks and security concerns had been considered.	Functional tool Document reporting the test made in a similar environment to the one that the potential customers would have.	Report of tool deployment in a customer application environment.

The tool provider must check each one of the levels to evaluate their current TRL. In addition, it is important to highlight that if the tool is already in the market, then it is in the last level of development, and from that point, TRL cannot be utilized to keep defining further development. If the tool evolves, then it may go back to a lower level of development due to new considerations in its design.

5.3. Tools catalogue enhanced with TRL arguments

The results from the adoption of the TRL model defined for HUBCAP brought successful results as each provider was able to better define their current level of development, provide a short description of this state of development, and additionally, generate further documentation that can be accessed and exploited by the stakeholders. As the TRL model is a new feature, not all the tool providers have yet been able to offer a complete documentation of their current TRL. As a summary, from all the tools included in the platform:

- 27% have a complete description and documentation on their TRL
- 73% have an incomplete definition of TRL that misses the proof documentation which is currently being developed to be included

After the implementation of the tool, new feedbacks were gathered. Some items from this feedback are mentioned here:

- Inclusion of additional details for the documents being uploaded into the platform. At the moment, the format is not specified as it can highly vary from company to company. The next development of the tool will imply the definition of a list of types of verification documents that can be utilized in the various stages of the TRL.
- Inclusion of the verification of TRL definition in the Quality Assurance (QA) process that is currently being designed in the T6.5. This with the intention to verify that the TRL is correctly defined and that the documents uploaded are coherent with the level specified. This due to the fact that even when specifications regarding the documents had been given, some partners do not comply with them, making necessary the implementation of a verification process.

6. Summary

This document reports on the deliverable D6.4, which consists of the version of contents that populate the models and tools catalogue on the HUBCAP platform. These contents count now more than one hundred assets among models and tools. In this document we focused on the enhancement of the MBD tools to validate their results and the new fields in the catalogues added to support TRL claims. In the next deliverables, the platform will be enriched further by guidelines and videos to help the user in using the tools and the platform itself.

References

[1] K. Namjoshi. Certifying model checkers. In proceedings of CAV, 2001
[2] A. Griggio, M. Roveri, S. Tonetta. Certifying proofs for SAT-based model checking. In Formal methods in system design, 2021.
[3] C. Barrett, R. Sebastiani, S. Seshia, Cesare Tinelli. Satisfiability Modulo Theories. Handbook of Satisfiability,2009.
[4] R. W. Floyd. Assigning meanings to programs. In Proceedings of Symposium on Applied Mathematics, number 32, 1967.
[5] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), *www.SMT-LIB.org*. 2016.
[6] J. C. Mankins: Technology Readiness Levels: A White Paper. (January 1995), From: http://www.hq.nasa.gov/office/codeq/trl/trl. (1995).
[7] M. Héder: From NASA to EU: The evolution of the TRL scale in Public Sector Innovation. Innov. J., 22 (2), 1–23 (2017).