Grant Agreement: 872698

Digital innovation HUBs and CollAborative Platform for cyber-physical systems

HUBCAP

The Second version of HUBCAP contents

Technical Note Number: D6.3

Version: 0.7

Date: April 2021

Public Document

http://hubcap.au.dk

Editors
Adrian Pop (RISE)

Contributors
Adrian Pop (RISE)
Claudio Sassanelli (POLIMI)
Pietro Braghieri (FBK)

Catalogues Support
Rosamaria Maniaci (ENGIT)

Sandboxing Middleware Support
Pietro Greco (ENGIT)

Reviewers
Mihai Neghina (ULBS)
Román Felipe Bastidas Santacruz (POLIMI)
Claudio Sassanelli (POLIMI)
Stefano Tonetta (FBK)
Peter Gorm Larsen (AU)

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	24-02-2021	Adrian Pop	Preliminary structure
0.2	03-03-2021	Pietro Braghieri	Added Tools API
0.3	08-03-2021	Adrian Pop	Integrate RISE and FBK changes
0.4	12-03-2021	Adrian Pop	Add the new tools and models
0.5	15-03-2021	Claudio Sassanelli Pietro Greco Adrian Pop	Control policies part, review, feedback
0.6	07-04-2021	Adrian Pop	Small fixes, references, acronyms
0.7	09-04-2021	Adrian Pop	Improvements via review

Abstract

The **HUBCAP Collaborative Platform** provides a **catalogue of models and tools** for Model-Based Design (MBD) for trial experiments to design and develop innovative CPS solutions with the support of MBD technology.

This deliverable (D6.3) consists of a second version of **tools** and **models** provided by HUBCAP partners that are uploaded, installed and tested inside the HUBCAP Sandboxing Middleware (HSM). In this version the platform and tools are enhanced to allow **multi-user collaboration**. In particular, access control policies will be tailored to the specific kind of modelling artefacts and analysis services supported by the platform. The catalogue will continuously be enriched along the project by the HUBCAP partners, together with the winners of the Open Calls, and in the future by the HUBCAP ecosystem.

D6.3 is defined as of **type "OTHER"**, given by the actual platform contents that are installed in the HSM. The catalogues can be browsed by accessing the platform.

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
UML	Unified Modeling Language
UAV	Unmanned Aerial Vehicle
VDM	Vienna Development Method
VM	Virtual Machine
VNC	Virtual Network Computing

Contents

1. Introduction

The **HUBCAP Collaborative Platform** aims to lower the Model-Based Design (MBD) entry barrier by providing **catalogues of models and tools** for MDB to support trial experiments that can design and develop innovative CPS solutions.

This deliverable (D6.3) consists of the second version of the HUBCAP Collaborative Platform and the tools and models provided by HUBCAP partners that are uploaded, installed and tested inside HSM. The deliverable focuses on multi-user collaboration.

2. Multi-user collaboration

2.1. Use-cases

Several multi-user collaboration use cases have been envisioned:

1. Multiple users collaborate via interactive screen sharing. This scenario can be used to solve problems or to teach a certain procedure, tool or functionality to a remote user. This is already supported by the HUBCAP platform as described in Section 2.2.
2. Users can use models and tools developed by service providers or other users. This is already supported by the HUBCAP platform via the tools/models catalogue and the sandbox functionality. See *D5.2 HUBCAP Collaboration Platform deliverable* [1].
3. Two or more users collaborate on the same model. A first prototype of a Modelica/FMI web-based editor supporting this feature is presented in Section 2.3.
4. Collaboration via tool integration. A proposal for an API that allows machine-to-machine interaction is given in Section 3.1.

2.2. Sandbox multi-user functionality

The HUBCAP platform has multi-user functionalities (more details are given in *D5.2 HUBCAP Collaboration Platform deliverable)*. These multi-user functionalities concern on how users can interact in HUBCAP, consider that any user can instantiate his/her own sandbox (so playing the owner role) and/or use the sandbox instantiated by another HSM owner that invited him (So playing the guest role, more details of roles given in the Section 3.2). There are three kinds of multi-user capabilities in the current HSM implementation:

- The owner of a Sandbox can invite guests. Multiple users of the same Sandbox can collaborate either (Figure 1 and Figure 2):
    - at tool level: sharing the tool's virtual screen, and the pair of virtual mouse and keyboard; or
    - at sandbox level: different users of the same Sandbox can simultaneously work on different tools of the Sandbox
- The tools within the Sandbox have access to the *Sandbox Shared Storage* (Figure 3)

These multi-user capabilities complement the access policy in the platform by allowing the users to interact in the platform ensuring the usability of tools and the security of the data utilized and obtained from it.

Figure 1. Sandbox Sharing Panel (in Control Panel) from which guests can be invited

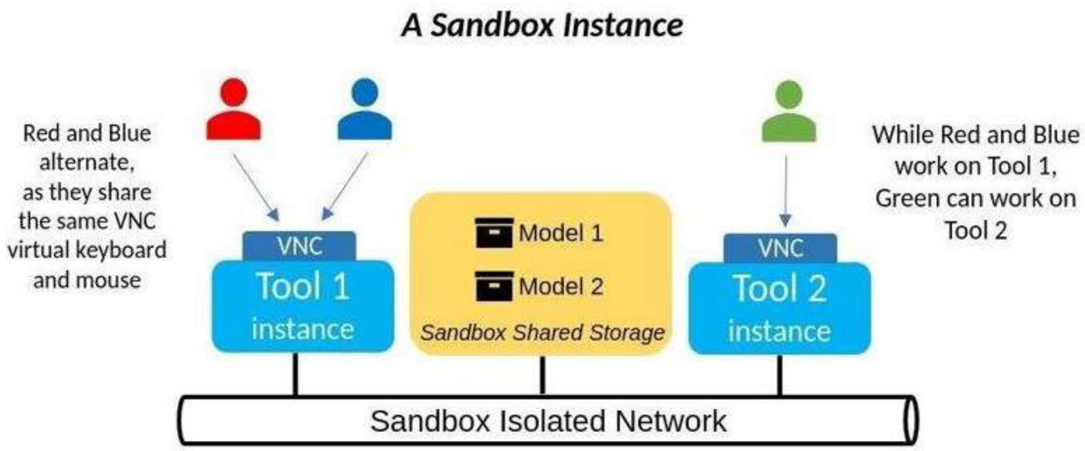


Figure 2. Typical collaboration scenario within the Sandbox.

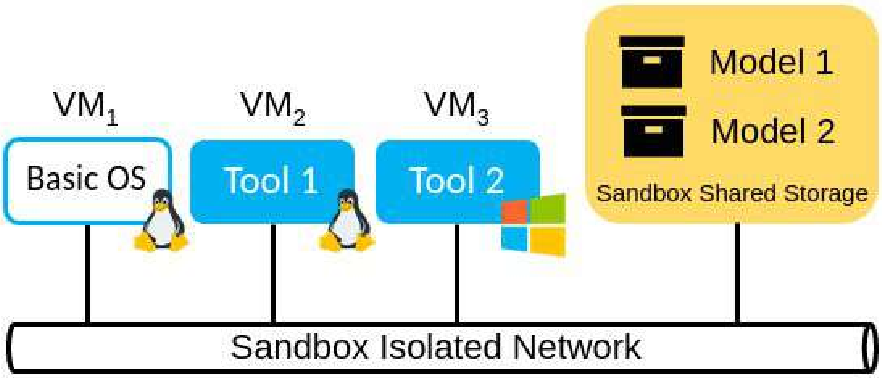


Figure 3. A Sandbox with a basic OS and two Tools. On the right the Sandbox Shared Storage

2.3. Collaboration in a web-based Modelica/FMI graphical editor

In the context of HUBCAP, RISE is developing a web-based graphical editor that allows users to develop Modelica models or FMI/SSP based composite-models via dragging-and-dropping and connecting components from model libraries or FMU sets directly in a web browser. Models can then be simulated remotely and the results displayed.

The prototype (Figure 4) is work in progress. Functionality has been developed for:

- Uploading and browsing Modelica libraries and FMU sets
- Drag & Drop & Connect components
- Simulate and display results

but it needs further integration to have a fully functioning prototype.

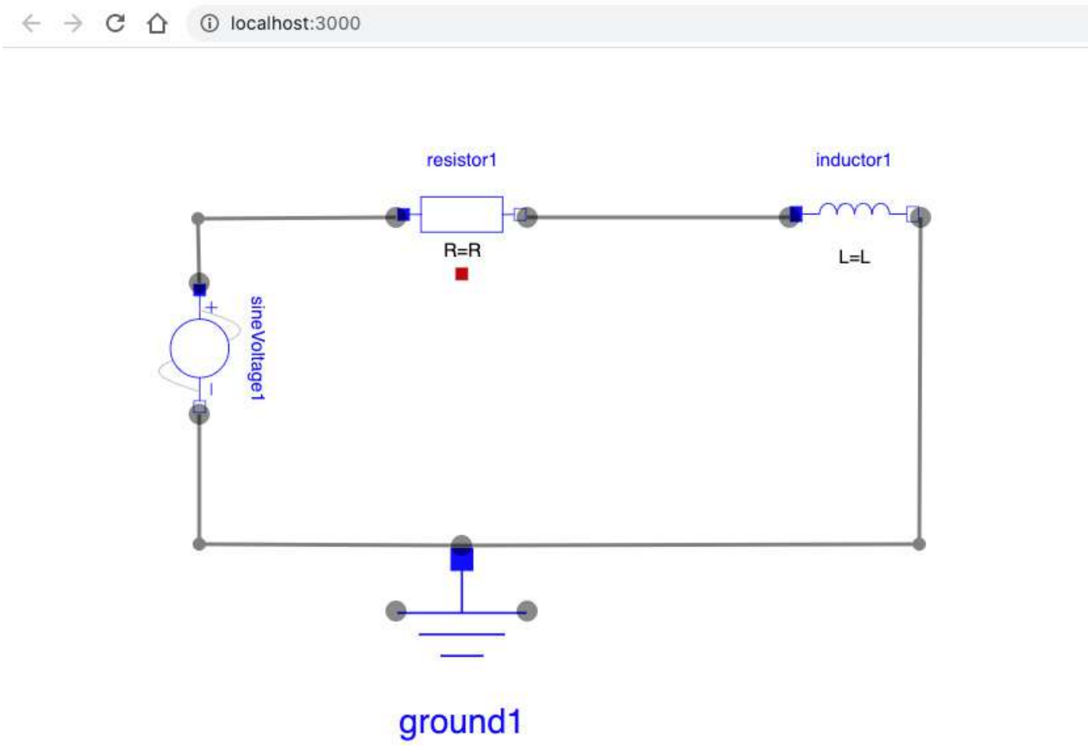


Figure 4. Prototype of a web based Modelica/FMI graphical editor

The design of the multi-user collaborative editing in the editor has been started. Several scenarios will be supported in the final version:

1. Sharing a model. A user logged into the platform can share his model and simulation results with other (already registered) users or via a link (URL) in an email. Models can be shared read-only or as a copy that can be modified.
2. Collaborative changes of the same model. Multi-users can edit any part of a model and the other users working on the model can be signalled of the changes by highlighting model

changes from other users. The user can then inspect the changes and accept or reject them if there are no conflicts. If there are conflicts the scenario below applies.

3. Handling model conflicts. When several users change the same model parts in a conflicting manner, conflict resolution is needed. Conflict resolution can be solved in several ways
   a) Users with conflicting changes can approve the changes from one user and abandon their own changes.
   b) An expert can be called to solve the conflict and the users will accept his/her decision.
   c) Users need to have a virtual meeting and discuss the conflicting changes to reach a resolution. Some functionality to facilitate such meetings is considered for implementation in the editor

3. Model and Tool integration

Collaboration in the HUBCAP platform can also be achieved via integration of models and tools. The current section focuses on this type of collaboration.

3.1. API for tools in the VMs

The set of tools provided by HUBCAP platform are, in terms of interaction, basically of two categories; those that allows the end user to interact via a human interface (UI or Command Line) and those that offer an API for the machine-to-machine interaction.

This section analyzes the second category, focusing on a possible standard API that allows for example a MDB tool to be integrated with back-end tools that support the model analysis such as the analysis of component-based specifications of system architectures or the analysis of finite-state systems or the safety assessment of systems.

Figure 5 depicts some possible machine-to-machine interactions; for example, we could have the MDB tool that runs at the end-user's home and interacts remotely with the analysis tool running in the hubcap platform; at the same time, the analysis tool could interoperate with a second analysis tool for some specific tasks.

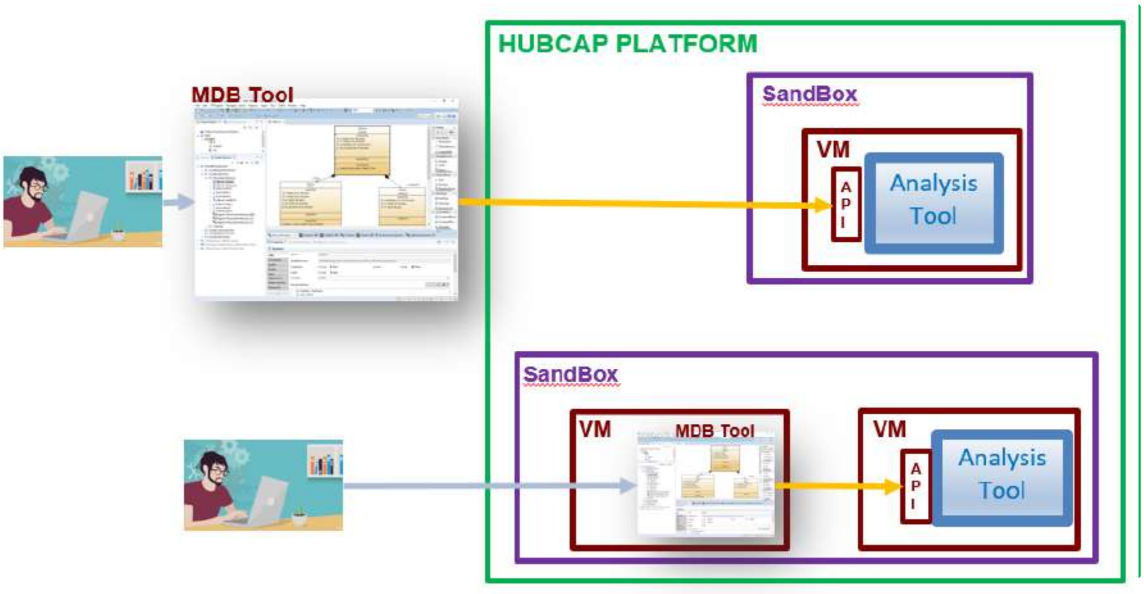


Figure 5. Platform architecture – interaction scenario

Tools running inside the Sandbox cannot (yet) expose APIs to the external world (i.e. to systems running outside the same Sandbox or the HSM). If a tool running inside the Sandbox acts as a server, it can **only** accept connections coming from the same sandbox. Therefore, APIs exposed by a tool in a Sandbox are only accessible to other tools running within the same Sandbox. A tool running inside the Sandbox can interact with an external system only if the connection is established **from** (starts from) the Sandbox (client runs in sandbox, server is outside the platform). Any other usage scenarios would need more support from HSM.

The proposed API is inspired by the solution already implemented in the FBK tools, based on the OASIS Open Services for Lifecycle Collaboration (OSLC) specifications which 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 [2] 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 (namely OCRA, nuXmv and xSAP) expose an API that allows the integration with any kind of client. Such an API extends the OSLC Automation specification that is defined by the OSLC Core Specification. The full specification is available at:

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

This specification builds on the OSLC Core Specification to define the resources and operations supported by an Open Services for Lifecycle Collaboration (OSLC) Automation provider. Automation resources define automation plans, automation requests and automation results of the software development, test, and deployment lifecycle. They represent individual resources as well as their relationships to other automation resources and to other linked resources outside of the automation domain. The intent of this specification is to define the set of HTTP-based RESTful interfaces in terms of HTTP methods: GET, POST, PUT and DELETE, HTTP response codes, mime type handling and resource formats.

The Automation resource properties are not limited to the ones defined in the specification; service providers may provide additional properties. In that way, FBK extended the automation resources by adding some specific properties that are described later in the document.

To better understand the API, it is useful to summarize the relationships between Automation Resources. Figure 5 shows such relationships:

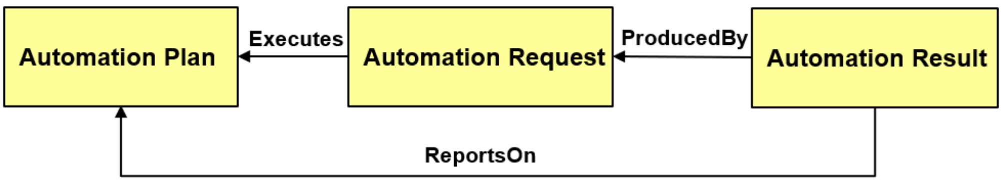


Figure 6. Relationships between automation resources

OSLC Automation allows not only automation of tasks and processes, but also an integration of any non-interactive tool without writing specific tool adapter (for example many command line tools).

For the analysis tools, it has been realized a specific OSLC Service Provider based on the OSLC Automation domain as depicted in Figure 6:

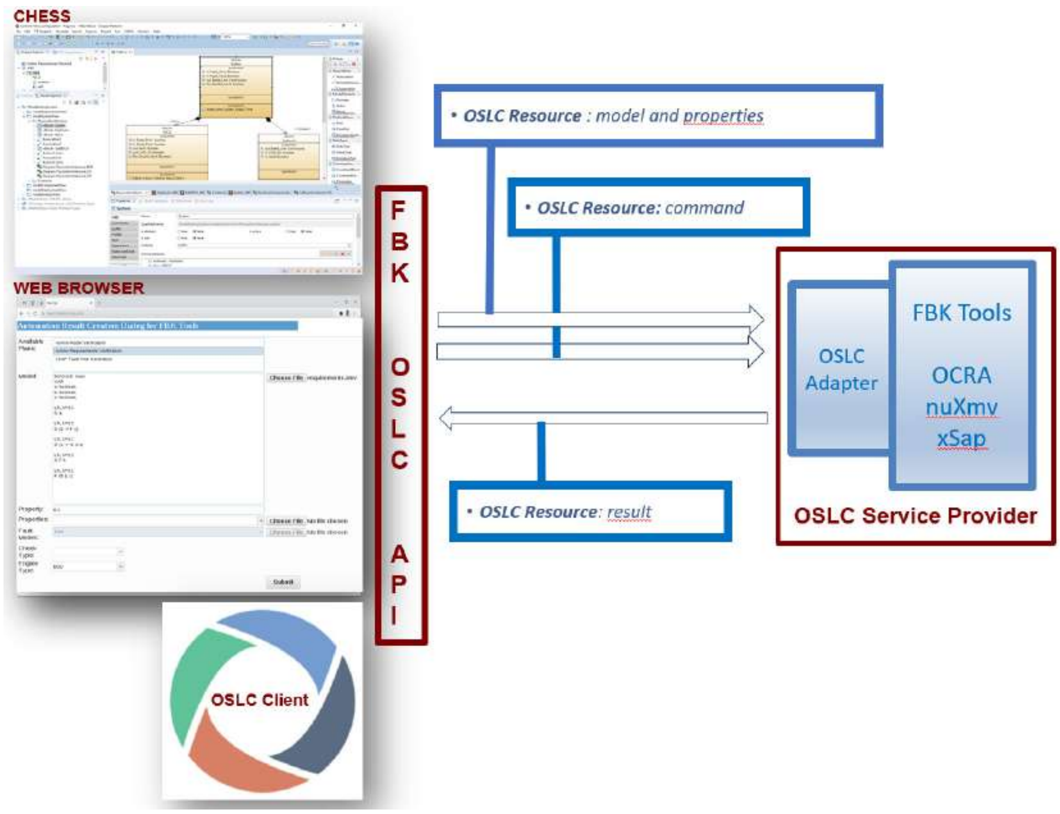


Figure 7. Architecture using the API

Figure 6 shows some tool clients (CHESS Eclipse, Web Browser and a generic OSLC client) that access through the OSLC API to the tool functionalities.

The analysis tool functionalities can be made available by means of Automation Plans; the title attribute <`dcterms:title`> of the Automation Plan selects the tool functionality. At the current time, the available Plans for the FBK analysis tools are the following:

Analysis Tool Functionality	Automation Plan Title
Check Contract Refinement	`ocra_check_refinement`
Check Contract Implementation	`ocra_check_implementation`
Check Contract Composite Implementation	`ocra_check_composite_impl`
Check Contract Validation Property	`ocra_check_validation_prop`
Compute Fault Tree	`ocra_compute_fault_tree`
Print System Implementation	`ocra_print_system_implementation`
Print Implementation Template	`ocra_print_implementation_template`
Instantiate Parametric Architecture	`ocra_instantiate_parametric_arch`
Get Required Architecture Parameters	`ocra_get_required_arch_parameters`
Behavior Model Check	`nuxmv_check_model`
Expand Fault Extensions	`xsap_expand_fault_extensions`
Extend Model	`xsap_extend_model`
Compute Fault Tree	`xsap_compute_fault_tree`
Compute FMEA Table	`xsap_compute_fmea_table`

For a more detailed description, the meaning of the analysis tool functionalities can be found in the FBK Tool Manuals.

The OSLC Service Provider exposes additional services for listing the service provider catalogue, the list of Automation Plans that are available in the current instance of service provider, the Automation Requests that were submitted and the list of Automation Results.

Each services has an end point that is listed in the following table:

OSLC Service Catalogue	<web_server_address>/oslc4j-registry/catalog

List of available Plans	<web_server_address>/<context>/services/autoPlans
List of Current Requests	<web_server_address>/<context>/services/autoRequests
List of Current Results	<web_server_address>/<context>/services/autoResults

For example, if we want to list the available Automation Plans we call the end point <web_server_address>/<context>/services/autoPlans and the corresponding answer will be a

RDF/XML output that lists all Automation Plans that are supported by the Service Provider. Following a fragment of such list:

```
▼<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:oslc_data="http://open-
 xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:oslc_auto="http://open-services.net/ns/auto#">
  ▼<oslc:ResponseInfo rdf:about="http://hubcap-oslc-es-tools.internal.cloudapp.net:80/eu.fbk.tools.oslc.provider/services/autoPlans">
     <oslc:totalCount rdf:datatype="http://www.w3.org/2001/XMLSchema#int">15</oslc:totalCount>
   ▼<rdfs:member>
     ▼<oslc_auto:AutomationPlan rdf:about="https://oslc-es-tools.fbk.eu/eu.fbk.tools.oslc.provider/services/autoPlans/7">
         <dcterms:description rdf:parseType="Literal">Print System Implementation</dcterms:description>
       ▶<oslc_auto:parameterDefinition>
         ...
         </oslc_auto:parameterDefinition>
         <dcterms:title rdf:parseType="Literal">ocra_print_system_implementation</dcterms:title>
       ▶<oslc_auto:parameterDefinition>
         ...
         </oslc_auto:parameterDefinition>
         <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2021-01-08T22:36:35.414Z</dcterms:modified>
       ▶<oslc_auto:parameterDefinition>
         ...
         </oslc_auto:parameterDefinition>
         <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2021-01-08T22:36:35.414Z</dcterms:created>
         <oslc:serviceProvider rdf:resource="http://oslc-es-tools.fbk.eu:80/oslc4j-registry/serviceProviders/2"/>
       ▶<oslc_auto:parameterDefinition>
         ...
         </oslc_auto:parameterDefinition>
         <dcterms:identifier>7</dcterms:identifier>
       ▶<oslc_auto:parameterDefinition>
         ...
         </oslc_auto:parameterDefinition>
       </oslc_auto:AutomationPlan>
     </rdfs:member>
   ▼<rdfs:member>
     ▶<oslc_auto:AutomationPlan rdf:about="https://oslc-es-tools.fbk.eu/eu.fbk.tools.oslc.provider/services/autoPlans/14">
```

The fragment shows that there is an Automation Plan associated to the function *OCRA Print System Implementation* (see <`dcterms:title`>) and that can be invoked by creating an Automation Request and linking the property <`oslc_auto:executesAutomationPlan`> to the Automation Plan Resource URL

https://oslc-es-tools.fbk.eu/eu.fbk.tools.oslc.provider/services/autoPlans/7

Every Automation Plan admits a set of input parameters that will be instantiated in the corresponding Automation Request. All admitted parameters of an Automation Plan are returned by the Service Provider sending a request on the Plan instance. For example, if the ID of the plan is 1 (see the Plan Catalogue for the whole ID list), the HTML response to the request

`<web_server_address>/eu.fbk.tools.oslc.provider/services/autoPlans/1` will be like the one shown in Figure 5:

OSLC Automation Plan

1: ocra_check_refinement

ID	**1**	
Title	**ocra_check_refinement**	
Description	**Checks the contract refinement of the contract base model**	
Parameters:		
	algorithmType	**[bmc, bdd, ic3, auto]**
	asynchronousExecution	**[true, false]**
	boundLength	
	contractModel	
	contractName	
	timeModel	**[discrete, hybrid]**
	timeout	

Figure 8. Element ocra_check_refinement

In Figure 5 are shown the ID of the plan (`1`), the name (`ocra_check_refinement`) and the command parameters (name and admitted values).

There is also a machine-to-machine representation of the Automation Plan that is shown by the following RDF/XML fragment:


```xml
<rdfs:member>
 <oslc_auto:AutomationPlan rdf:about="https://oslc-es-tools.fbk.eu/eu.fbk.tools.oslc.provider/services/autoPlans/1">
  <oslc_auto:parameterDefinition>
   <oslc:Property>
    <oslc:defaultValue rdf:datatype="http://www.w3.org/2001/XMLSchema#int">300</oslc:defaultValue>
    <oslc:name>timeout</oslc:name>
    <oslc:occurs rdf:resource="http://open-services.net/ns/core#Exactly-one"/>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <dcterms:title rdf:parseType="Literal">Execution Timeout</dcterms:title>
    <dcterms:description rdf:parseType="Literal">The tool execution timeout (seconds)</dcterms:description>
   </oslc:Property>
  </oslc_auto:parameterDefinition>
  <oslc_auto:parameterDefinition>
   <oslc:Property>
    <oslc:allowedValue>false</oslc:allowedValue>
    <oslc:allowedValue>true</oslc:allowedValue>
    <oslc:defaultValue rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</oslc:defaultValue>
    <oslc:name>asynchronousExecution</oslc:name>
    <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
    <dcterms:title rdf:parseType="Literal">Asynchronous Execution </dcterms:title>
    <dcterms:description rdf:parseType="Literal">If true the tool execution will be performed asynchronously</dcterms:description>
   </oslc:Property>
  </oslc_auto:parameterDefinition>
  <oslc_auto:parameterDefinition>
   <oslc:Property>
    <oslc:allowedValue>auto</oslc:allowedValue>
    <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
    <oslc:name>algorithmType</oslc:name>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <oslc:allowedValue>ic3</oslc:allowedValue>
    <dcterms:title rdf:parseType="Literal">Algorithm Type</dcterms:title>
    <dcterms:description rdf:parseType="Literal">Type of algorithm</dcterms:description>
    <oslc:defaultValue>auto</oslc:defaultValue>
    <oslc:allowedValue>bmc</oslc:allowedValue>
    <oslc:allowedValue>bdd</oslc:allowedValue>
   </oslc:Property>
  </oslc_auto:parameterDefinition>
  <oslc_auto:parameterDefinition>
   <oslc:Property>
    <oslc:allowedValue>hybrid</oslc:allowedValue>
    <oslc:allowedValue>discrete</oslc:allowedValue>
    <oslc:defaultValue>discrete</oslc:defaultValue>
    <oslc:name>timeModel</oslc:name>
    <oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one"/>
    <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <dcterms:title rdf:parseType="Literal">Time Model</dcterms:title>
    <dcterms:description rdf:parseType="Literal">The type of time model (discrete or hybrid)</dcterms:description>
   </oslc:Property>
  </oslc_auto:parameterDefinition>
```

The highlighted items are the definition of some request parameters, for example the `algorithmType` and the `timeModel` and the related types/allowed values.

Execution Steps

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

1. Find in the catalogue the URI of the Automation Plan that maps the required tool function.
2. Create an Automation Request that links such a Plan, set the input parameters (considering the Automation Plan description for their definitions) and submit the Request to the OSLC Service Provider.
3. Once the Request has been submitted, its status can be "in progress" or "completed" depending on the request being synchronous or asynchronous.
4. If it is synchronous, when the Request is completed, the Service Provider can be asked for the Automation Result that is linked to the Request. (See the Automation Result property `<oslc_auto:producedByAutomationRequest>`)
5. If the request is asynchronous, the Service Provider can be polled by asking the status of the Request (see property `<oslc_auto:state>`) and, if completed, the Service Provider can eb asked for the Automation Result that is linked to the Request.

3.2. Control policies

According to the NIST (National Institute of Standards and Technology) [2], access control policies are the highest abstraction of the control systems and has the function to protect users, information and deployers. It defines how access is managed and who can have access to information under certain circumstances across the organization.

One of the HUBCAP's main fundamental management responsibilities is the security assurance for the companies and HUBs that interact in the platform. For this, as in any IT system that deals with sensitive or important data, access (authentication) controls had been included in the platform. In addition to the authentication mechanism that the system has, these access controls are centred in determining the activities and permissions that different users can perform or have and how they are structured.

The simplest way to define this concept is by dividing it in its three main components [2]:

- **Identification:** the most basic identification method which simply identifies an individual by classifying it in a group of users. This is usually done through an PGP of an e-mail address, a username, etc. In HUBCAP platform, the identification level is made through an e-mail address linked to the user that could be a service consumer or provider.
- **Authentication:** This level ensures that the identity of the user is authentic and is being used by the right individual. In HUBCAP, this level is presented as the validation of a password linked to the username.
- **Authorization:** It encompasses the set of actions that which user is allowed to perform depending on its identity. In HUBCAP, the authorization level is translated in the actions that each user can execute in the sandbox middleware.

In a nutshell, access controls are utilized to authenticate and define the activities that each individual within the organization can perform. Access control policies can be presented in different ways, for the HUBCAP platform, it was defined that Role-Based Access Control (RBAC) is the most feasible policy to ensure protection users, their information and the platform itself.

3.2.1. Role-Based Access Control (RBAC) policy applied to HUBCAP

RBAC [2] is part of the non-discretionary access control policies. These policies have rules that are not established at the discretion of the individual in the platform, the controls defined in them are not modifiable by users, they can be only modified through administrative interventions.

This policy, as its name states, grants permissions to users through roles that are based on certain competencies and responsibilities that each user has in the HUBCAP environment. The actions that each user is able to execute are based on their role, this simplifies the management of permissions by only subscribing or unsubscribing users to different roles depending on the requirements without the need to update privileges individually for each user. Additionally, the utilization of roles allows also to modify the actions or operations that the users subscribed to different roles can perform, actions and operations can be added, erased or modified.

To define the access control policy for the HUBCAP platform, the identification of different roles in the system was the main task to perform. After this, each specific role was based on the assignation of the corresponding permissions to use the resources available in the HSM. These permissions were defined to appropriately fit the needs of each user across the platform.

In the RBAC model taxonomy, there are basic models that define it. For the HSM, four of them are highly relevant to be included. The first, *Core RBAC,* includes the basic set of features that are present in any RBAC and make this method different from other access policies methodologies. The second, *Hierarchical RBAC,* adds the concept of hierarchy to the model and additionally includes an extra layer to cluster the roles. The third, *statistically constrained RBAC,* creates constrains to some roles in order to avoid the possibility of data leaking and at the same time secure the information with which companies interact in the platform. Finally, *Dynamic constrained RBAC* imposes constrains in time on the activities that some roles can perform. These four models are partially implemented in the middleware and are intended to be further implemented in time with the growth of the platforms' features and size. Hereafter, a more detailed definition on the models is made:

Core RBAC: It includes five basic administrative sets that this policy has: users, roles, permission, operations and objects. Roles are the main definition of this model; permissions are associated with roles and then users can be made members of roles depending on the requirements.

Hierarchical RBAC: In this model, it is defined how roles can have overlapping responsibilities and rights. Some users that belong to different roles can require the same operations, in this context, the assignation of permissions can be layered hierarchically, making the rights assignation process more efficient.

Statically constrained RBAC: it avoids the need of constant access monitoring to each user by creating restriction of access to data or sandboxes depending on the user. In this way, companies utilizing the services of the platform can only access to their own sandboxes and interact only with them as owner or guest. In the same way, HUBS can interact, if necessary, not only with their own tools and models but also instantiate Sandboxes containing Tools and Models provided by other HUBs. This increases the interoperability characteristic in the platform and eases the construction of collaborative models or tools. In HUBCAP, the statical constrain is defined through profiles.

Dynamically constrained RBAC: dynamical constrains are defined to regulate the operations that a member of the role can perform. This is made by granting or retiring the user membership to roles. Some roles are occupied by users such as companies at certain period of times where they contract the service in the middleware sandbox of HUBCAP.

3.2.2. HUBCAP access control policy

As it was mentioned in the previous section, in the HUBCAP Sandboxing Middleware a RBAC policy is implemented. To have a complete RBAC policy, the first two components (identification and authentication) of the access control policies must be included in it. For this, as it was stated in the deliverable 5.2, the users must execute the two-factors authentication. First, the users authenticate their access through the DIHIWARE platform, and then he will be able to receive a password to access the Sandboxing Middleware Web Application using the Apache Basic Authentication. The password is defined with certain security parameters:

- long
- randomly generated by the operating system for every user's session
- sent automatically to the user's private e-mail
- destroyed automatically after a short period of time

After this process, in the Sandboxing Middleware, users are grouped in in two main classes (Profiles and Roles) in order to have more control over their allowed activities. That is another contribution to Sandbox hardening. At this point core and hierarchical RBAC are implemented for the users. This ensures that certain individuals that access the platform have the correspondent rights to interact with it seamlessly. Furthermore, profiles are statically constrained with each user or company, defining each area of the middleware that it has access to. Finally, the roles can be attached dynamically to a sandboxing middleware user and define the relationships between the user and the sandboxes thus determining to which functionalities of a specific sandbox instance a user can access. Is important to highlight that these permissions can vary on time, creating the need of the utilization of this dynamic constrain RBAC.

3.2.2.1. HUBCAP profiles and roles

For the third component of the access control policy, the authorization, in HUBCAP the profiles are constructed over the core RBAC model and organized hierarchically as it can be observed in the Figure 5. In it, the profiles are divided in 3 categories organized from left to right decreasing the functionalities/rights to which the user can have access to.

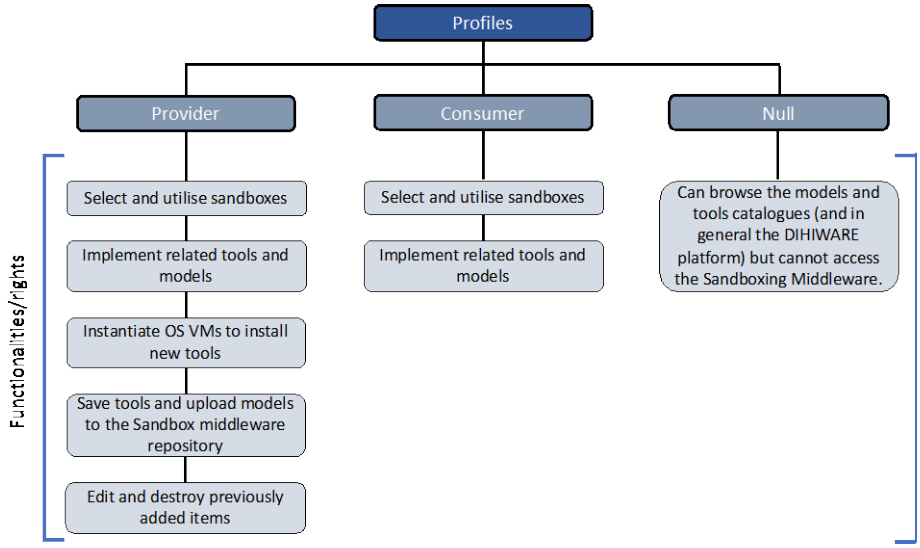


Figure 9. Hierarchically ordered profiles

In addition to the profiles, two roles are defined as part of the HUBCAP environment and that can be assigned to the profiles.

Role	Owner	Guest
Description	is a Sandboxing Middleware user that instantiates a new Sandbox. As the owner, he can: • Destroy his own Sandbox • Share it with other Sandboxing Middleware users (who become his guests) • Upload or download local archives to/from the Sandbox.	Is a Sandboxing Middleware user invited to access one or more Sandboxes which they are not an owner of. Owner and Guests of the same Sandbox can collaborate with each other sharing the sandbox's screen, mouse and keyboard.

The definition of these roles is basic for the platform operation, nevertheless the option of adding new ones is constantly being explored as further functionalities are added.

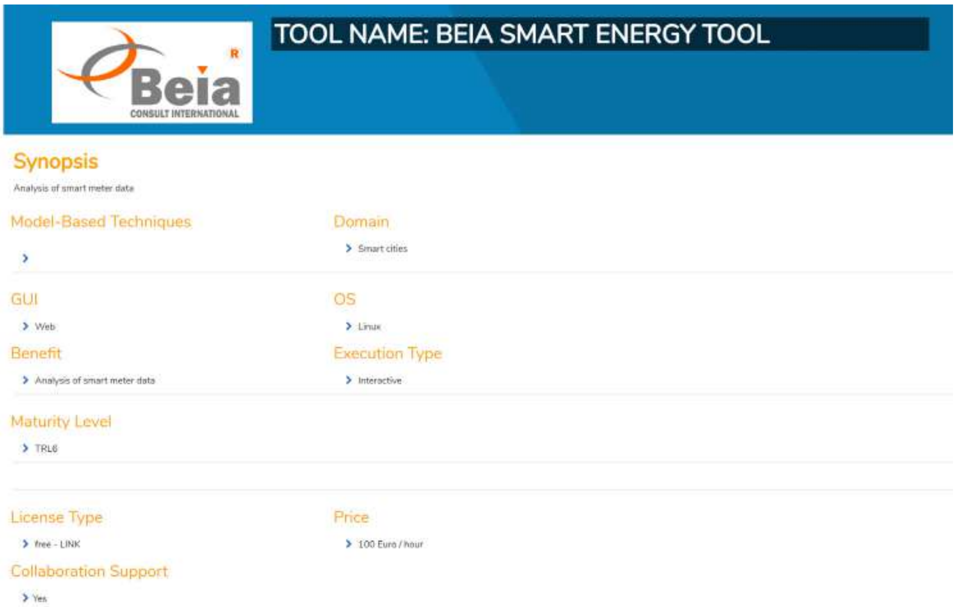
4 New Tools and Models

This section presents the new tools and models added to the HUBCAP platform catalogues since deliverable *D6.2 The Initial HUBCAP Model-Based Services* [3].

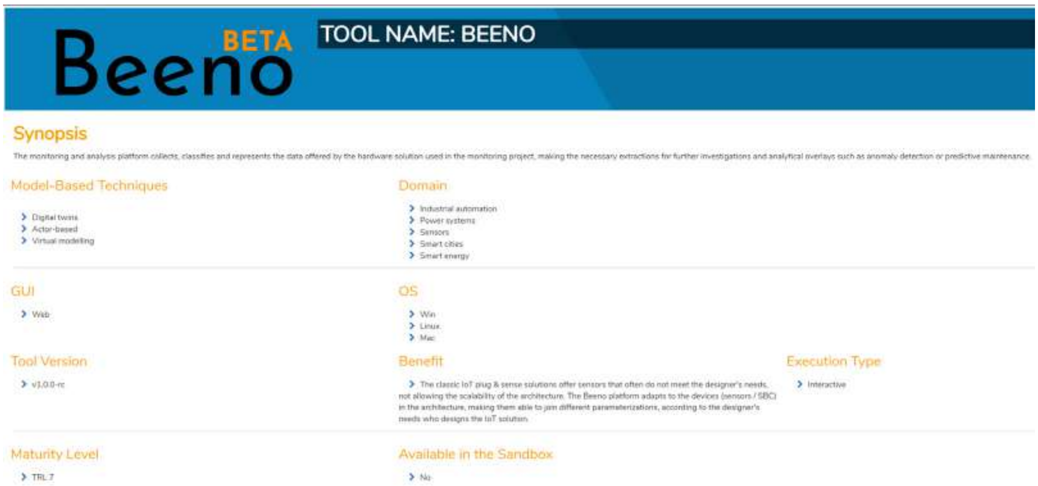
4.1 New Tools

All tools can be found in the HUBCAP platform tools catalogue here:
https://hubcap-portal.eng.it/group/guest/tools-catalogue

4.1.1. Beia Smart Energy Tool



4.1.2. Beeno



Beeno BETA

TOOL NAME: BEENO

Synopsis

The monitoring and analysis platform collects, classifies and represents the data offered by the hardware solution used in the monitoring project, making the necessary extractions for further investigations and analytical overlays such as anomaly detection or predictive maintenance.

Model-Based Techniques

- Digital twins
- Actor-based
- Virtual modelling

Domain

- Industrial automation
- Power systems
- Sensors
- Smart cities
- Smart energy

GUI

- Web

OS

- Win
- Linux
- Mac

Tool Version

- v1.0.0-rc

Benefit

- The classic IoT plug & sense solutions offer sensors that often do not meet the designer's needs, not allowing the scalability of the architecture. The Beeno platform adapts to the devices (sensors / SBC) in the architecture, making them able to join different parameterizations, according to the designer's needs who designs the IoT solution.

Execution Type

- Interactive

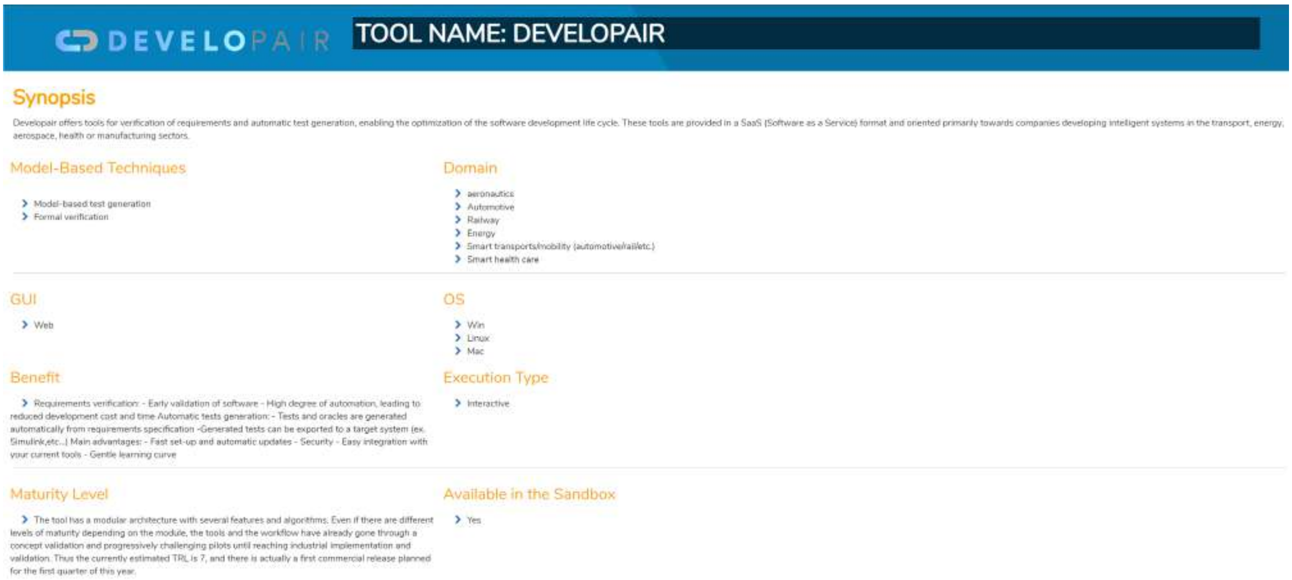
Maturity Level

- TRL 7

Available in the Sandbox

- No

4.1.3. DEVELOPAIR



DEVELOPAIR

TOOL NAME: DEVELOPAIR

Synopsis

Developair offers tools for verification of requirements and automatic test generation, enabling the optimization of the software development life cycle. These tools are provided in a SaaS (Software as a Service) format and oriented primarily towards companies developing intelligent systems in the transport, energy, aerospace, health or manufacturing sectors.

Model-Based Techniques

- Model-based test generation
- Formal verification

Domain

- aeronautics
- Automotive
- Railway
- Energy
- Smart transports/mobility (automotive/rail/etc.)
- Smart health care

GUI

- Web

OS

- Win
- Linux
- Mac

Benefit

- Requirements verification: - Early validation of software - High degree of automation, leading to reduced development cost and time Automatic tests generation: - Tests and oracles are generated automatically from requirements specification -Generated tests can be exported to a target system (ex. Simulink,etc...) Main advantages: - Fast set-up and automatic updates - Security - Easy integration with your current tools - Gentle learning curve

Execution Type

- Interactive

Maturity Level

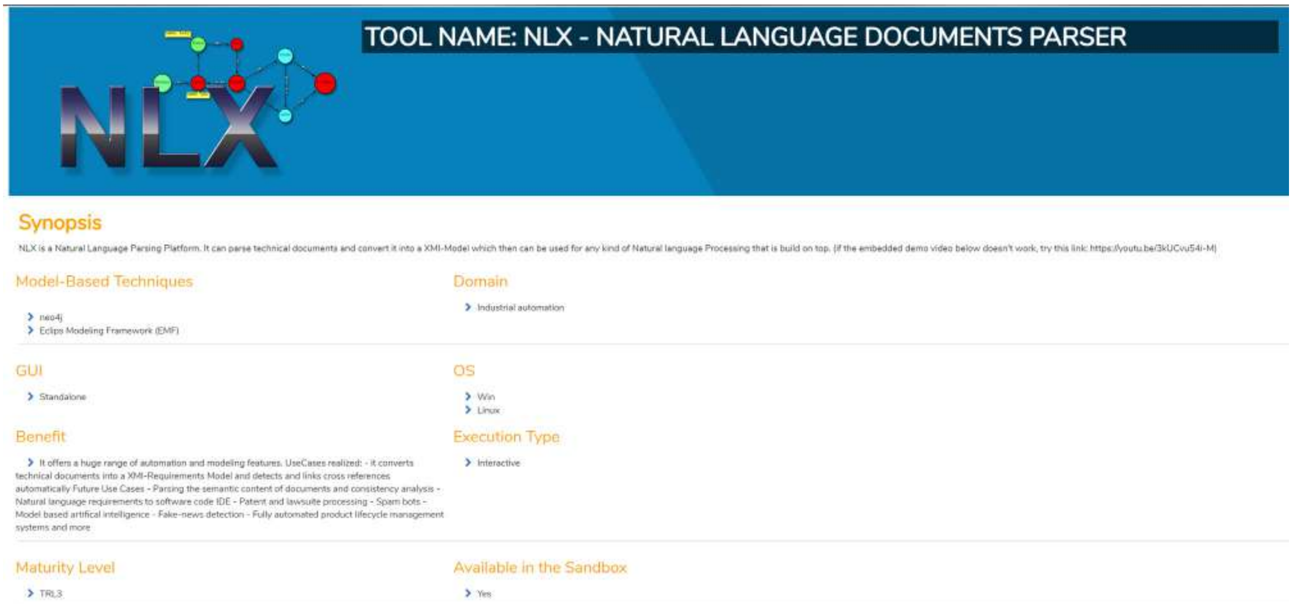
- The tool has a modular architecture with several features and algorithms. Even if there are different levels of maturity depending on the module, the tools and the workflow have already gone through a concept validation and progressively challenging pilots until reaching industrial implementation and validation. Thus the currently estimated TRL is 7, and there is actually a first commercial release planned for the first quarter of this year.

Available in the Sandbox

- Yes

4.1.4. Evitado System Simulator

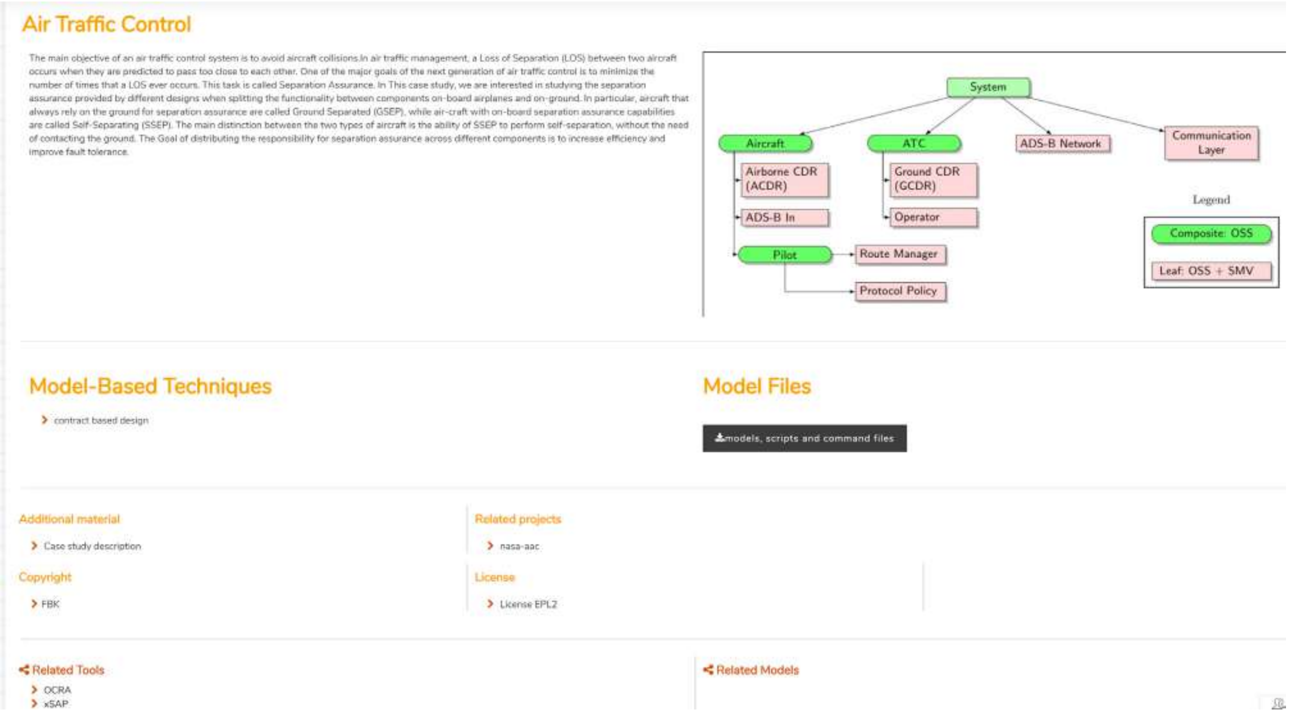
4.1.5. NLX – Natural Language Documents Parser



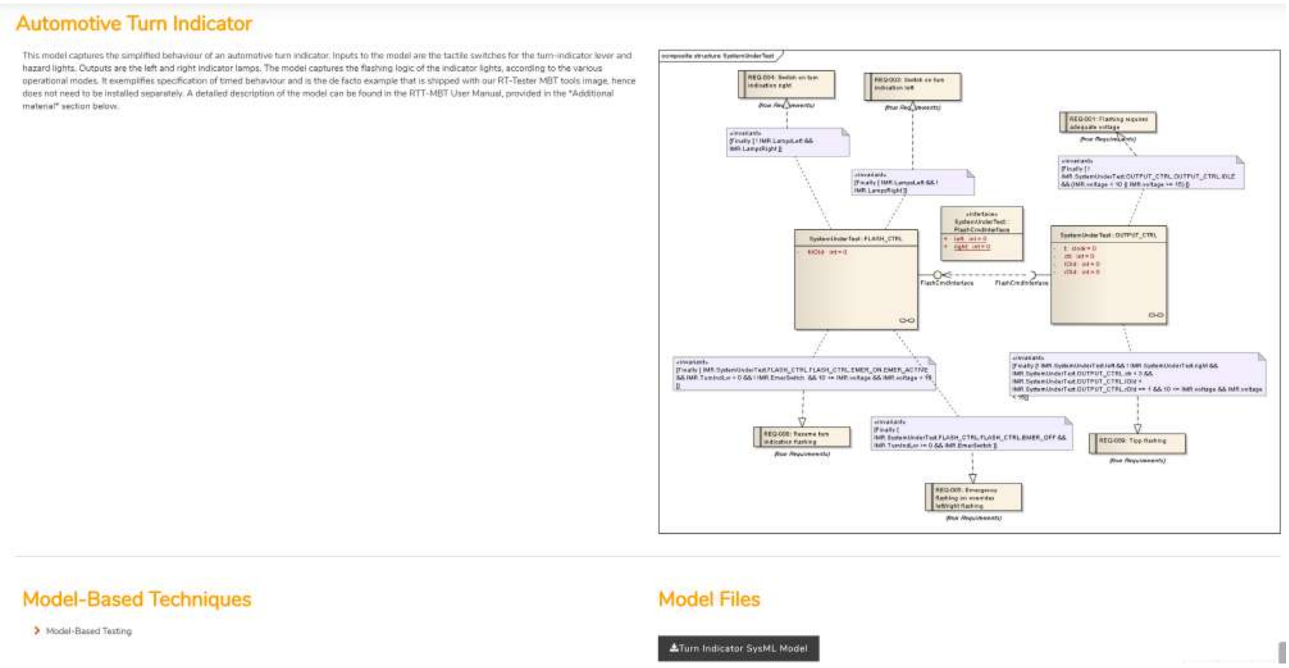
4.2. New Case Study Models

All models can be found in the HUBCAP platform catalogue here:
https://hubcap-portal.eng.it/group/guest/models-catalogue

4.2.1. Air Traffic Control



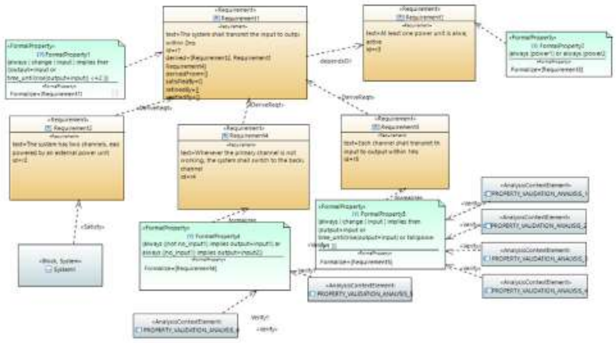
4.2.2. Automotive Turn Indicator



4.2.3. Dual Channel

Dual Channel

The model shows an example of dual channel design. The requirements are formalized and structured into contracts. The formal properties are validated with formal techniques.



Model-Based Techniques

- Requirements formal analysis
- Contract-based design

Model Files

4.2.4. Ferryman

Ferryman

In this example it is shown the formalization of the Ferryman problem in NuSMV. The counterexample of the formula provides a solution to the problem.

Model-Based Techniques

- model checking

Model Files

Model file

Additional material
- Ferryman problem

Related projects

Copyright
- FBK

Available in the Sandbox
- No

License
- License EPL2

Purpose
- example to get started

Cost
- This model is a simple example on how to model simple problems using NuSMV

Application Domain
- other

Target User
- Newbie

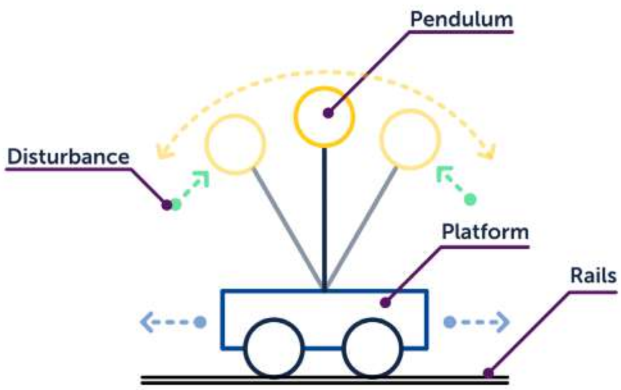
Related Tools
- nuXmv

Related Models

4.2.5. Inverted Pendulum

Inverted Pendulum

This model is an electronically controlled inverted pendulum that demonstrates the co-simulation capabilities of AutoFOCUS 3 and how this feature can be leveraged for tool-interoperability. For the tool interaction, we use the Functional Mock-up Interface (FMI) standard: The pendulum's dynamics and the PID controller are modeled using OpenModelica, whereas the input and environment models have been developed in AutoFOCUS 3 and Overture - both representing models with a discrete notion of time. The system to be simulated is an inverted pendulum mounted on a platform on rails. The pendulum's movement is parallel to the platform's tracks. Moreover, the inclination of the pendulum is assumed to be measurable. This results in a controllable system. The goal of the simulated system is not only to stabilize the pendulum given some disturbances acting on the pendulum. In addition, the inclination of the pendulum shall be able to be controlled by an input signal. The environment model responsible for simulating the disturbances acting on the pendulum is developed in Overture by means of the Vienna Development Method (VDM). The input model providing the desired inclination of the pendulum over time is developed using AutoFOCUS 3, relying on the stream-processing semantics of the FOCUS theory. Finally, the system model including the dynamics of the inverted pendulum as well as the PID controller are modeled in OpenModelica leveraging the continuous semantics of the Modelica language and the vast libraries of the tool.



Model-Based Techniques

- Component-Based Design
- Co-Simulation

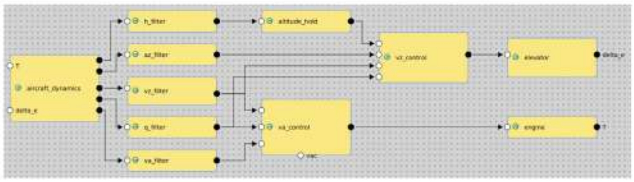
Model Files

Inverted Pendulum

4.2.6. Rosace

ROSACE

The model represents a set of software-defined functions of a baseline flight-controller whose components shall be deloyed on a platform consisting of two many-core tiles. In contrast to the original model, different periods of the tasks implementing the components are not considered and the same period is assumed for all tasks.



Model-Based Techniques

- Component-based Design

Model Files

Rosace

Model Version
- 2020-10

Available in the Sandbox
- Yes

Application Domain
-

Additional material
- Rosace Flight Controller Description

License
- APACHE 2.0

Related projects
-

Related Tools
- AutoFOCUS3

Related Models

4.2.7. Sense Spacecraft Rate

Sense SpacecraftRate

A simple example of a model with sensors. This model provides a simple example of refinement verification in OCRA. The model is composed by 2 sensors, 2 monitors and a Selector.

SenseSpacecraftRate_singlefailure.oss

```
#include "symbols.h"

COMPONENT SenseSpacecraftSpeed system
 INTERFACE
  INPUT PORT speed: continuous;
  INPUT PORT fail1: boolean;
  INPUT PORT fail2: boolean;
  OUTPUT PORT sensed_speed: real;
  OUTPUT PORT sensed_speed_is_present: boolean;

  CONTRACT sense
   assume: not fail1 and not fail2 and always ((not fail1) or (not fail2));
   guarantee:
    always (macro and
        sensed_speed_is_present);

 REFINEMENT
  SUB sensor1: SpeedSensor;
  SUB sensor2: SpeedSensor;
  SUB monitor1: MonitorPresence;
  SUB monitor2: MonitorPresence;
  SUB selector: Selector;

  CONNECTION sensor1.speed := speed;
  CONNECTION sensor1.fail := fail1;
  CONNECTION sensor2.speed := speed;
  CONNECTION sensor2.fail := fail2;
  CONNECTION monitor1.input_is_present := sensor1.sensed_speed_is_present;
  CONNECTION monitor2.input_is_present := sensor2.sensed_speed_is_present;
  CONNECTION monitor1.enabled := (selector.current_use=1);
  CONNECTION monitor2.enabled := (selector.current_use=2);
  CONNECTION selector.input1 := sensor1.sensed_speed;
  CONNECTION selector.input1_is_present := sensor1.sensed_speed_is_present;
  CONNECTION selector.input2 := sensor2.sensed_speed;
  CONNECTION selector.input2_is_present := sensor2.sensed_speed_is_present;
  CONNECTION selector.switch_current_use := monitor1.absence_alarm or
                              monitor2.absence_alarm;
  CONNECTION sensed_speed := selector.output;
  CONNECTION sensed_speed_is_present := selector.output_is_present;

  CONTRACT sense REFINEDBY sensor1.sense, sensor2.sense,
            monitor1.monitor, monitor2.monitor,
            selector.select, selector.switch;

COMPONENT SpeedSensor
 INTERFACE
  INPUT PORT speed: continuous;
  INPUT PORT fail: boolean;
  OUTPUT PORT sensed_speed: real;
  OUTPUT PORT sensed_speed_is_present: boolean;

  CONTRACT sense
   assume: TRUE;
   guarantee:
    always ((sensed_speed_is_present implies
            ((sensed_speed - speed <= error) and
          (sensed_speed - speed >= - error))) and
        ((not fail) implies sensed_speed_is_present));
```

4.2.8. Timed Fischer

Timed Fischer

Implementation and verification of Fischer protocol in timed nuXmv.

```
procedure FISCHER(pid, c, id)
    loop
        while id ≠ 0 do
            skip
        x ← random(0, c)
        wait_at_most(c)
        id ← pid
        wait_at_least(c)
        if id = pid then
            Critical Section
            id ← 0
```

Model-Based Techniques

- Model checking

Model Files

Models and cmd files

Additional material

- Timed nuXmv paper

Related projects

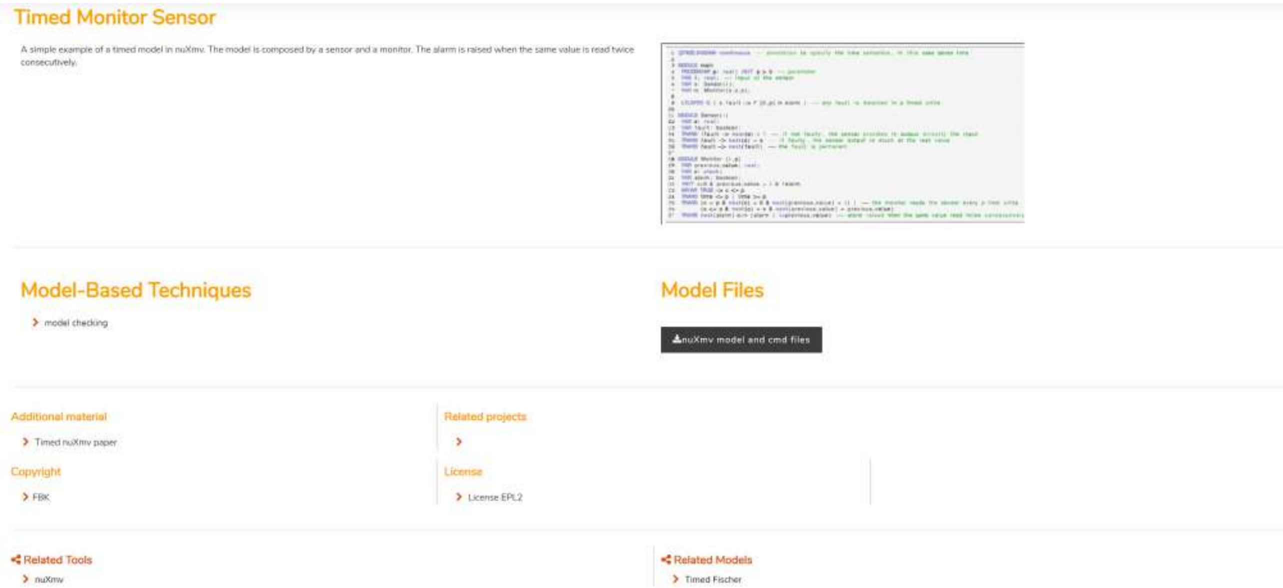
Copyright

- FBK

License

- License EPL2

4.2.9. Timed Monitor Sensor



5. Summary

This document reports on the deliverable D6.3, which consists of the second version of platform and the models / tools that populate the catalogue. These models and tools are provided by the HUBCAP partners, and they are installed and tested in the platform HSM. The focus of this deliverable is existing and planned multi-user functionality and tool integration on the HUBCAP platform.

References

[1] HUBCAP, *D5.2 HUBCAP Collaboration Platform deliverable*, https://www.hubcap.eu/project-details, 2020.

[2] Open OASIS, *OSLC Specifications,* https://open-services.net/specifications/, Accessed April 2021.

[3] V. Hu, D. Ferraiolo og R. Kuhn, *NIST 7316 - Assessment of access control systems,* NIST, 2006.

[4] HUBCAP, *D6.2 The Initial HUBCAP Model-Based Services,* https://www.hubcap.eu/project-details, 2020.

Appendix A – FBK Tools Automation Plans

Here are listed the current Automation Plans that maps the FBK Tool Functions.

OCRA Check Contract Refinement

O **OSLC Automation Plan**

1: ocra_check_refinement

ID	**1**	
Title	**ocra_check_refinement**	
Description	**Checks the contract refinement of the contract base model**	
Parameters:		
	algorithmType	**[bmc, bdd, ic3, auto]**
	asynchronousExecution	**[true, false]**
	boundLength	
	contractModel	
	contractName	
	timeModel	**[discrete, hybrid]**
	timeout	

OCRA Check Contract Implementation

O **OSLC Automation Plan**

2: ocra_check_implementation

ID	2	
Title	ocra_check_implementation	
Description	Verifies if a behaviour model satisfies the contracts defined in the contract based model	
Parameters:		
	algorithmType	[bmc, bdd, ic3, auto]
	asynchronousExecution	[true, false]
	behaviourModel	
	boundLength	
	componentName	
	contractModel	
	contractName	
	oldModelFormat	
	timeModel	[discrete, hybrid]
	timeout	

OCRA Check Contract Composite Implementation

O **OSLC Automation Plan**

3: ocra_check_composite_impl

ID	3	
Title	ocra_check_composite_impl	
Description	Verifies if a composition of behaviour models satisfie the contracts defined in the contract based model	
Parameters:		
	algorithmType	[bmc, bdd, ic3, auto]
	asynchronousExecution	[true, false]
	boundLength	
	componentsMapping	
	contractModel	
	oldModelFormat	
	skipRefinementCheck	
	timeModel	[discrete, hybrid]
	timeout	

OCRA Compute Fault Tree

O **OSLC Automation Plan**

4: ocra_compute_fault_tree

ID	**4**	
Title	**ocra_compute_fault_tree**	
Description	**Generates a hierarchical fault tree from a contract based system decomposition**	
Parameters:		
	asynchronousExecution	**[true, false]**
	contractModel	
	contractName	
	timeModel	**[discrete, hybrid]**
	timeout	

OCRA Check Validation Properties

O **OSLC Automation Plan**

5: ocra_check_validation_prop

ID	**5**	
Title	**ocra_check_validation_prop**	
Description	**Check validation properties against the contract based model**	
Parameters:		
	algorithmType	**[bmc, bdd, ic3, auto]**
	asynchronousExecution	**[true, false]**
	componentName	
	contractModel	
	possibility	
	properties	
	propertyName	
	propertyValidationType	**[consistency, possibility, entailment]**
	timeModel	**[discrete, hybrid]**
	timeout	
	wholeArchitecture	

OCRA Print System Implementation

O **OSLC Automation Plan**

6: ocra_print_system_implementation

ID	6	
Title	ocra_print_system_implementation	
Description	Compute a system implementation of a contract based mdoel	
Parameters:		
	asynchronousExecution	[true, false]
	componentsMapping	
	contractModel	
	timeModel	[discrete, hybrid]
	timeout	

OCRA Print Implementation Template

O **OSLC Automation Plan**

8: ocra_print_implementation_template

ID	8	
Title	ocra_print_implementation_template	
Description	Print implementation template	
Parameters:		
	asynchronousExecution	[true, false]
	componentName	
	componentsMapping	
	contractModel	
	oldModelFormat	
	timeout	

OCRA Instantiate Parametric Architecture

O OSLC Automation Plan

9: ocra_instantiate_parametric_arch

ID	**9**	
Title	**ocra_instantiate_parametric_arch**	
Description	**Instantiate the parameterized architecture**	
Parameters:		
	architectureParameters	
	asynchronousExecution	**[true, false]**
	contractModel	
	timeout	

OCRA Get Required Architecture Parameters

O OSLC Automation Plan

10: ocra_get_required_arch_params

ID	**10**	
Title	**ocra_get_required_arch_params**	
Description	**Get the parameters of the parameterized architecture**	
Parameters:		
	asynchronousExecution	**[true, false]**
	boundLength	
	contractModel	
	timeout	

nuXmv Check Model

O OSLC Automation Plan

11: nuxmv_check_model

ID	**11**	
Title	**nuxmv_check_model**	
Description	**Perform the model checking of the behaviour model against properties**	
Parameters:		
	algorithmType	**[bmc, bdd, ic3, klive]**
	asynchronousExecution	**[true, false]**
	behaviourModel	
	checkType	**[invar, ltlspec, ctlspec]**
	properties	
	timeout	

xSap Expand Fault Extensions

O OSLC Automation Plan

12: expand_fault_extensions

ID	**12**	
Title	**expand_fault_extensions**	
Description	**Expand fault extension**	
Parameters:		
	asynchronousExecution	**[true, false]**
	faultExtensions	
	timeout	

xSap Extend Model

O **OSLC Automation Plan**

13: xsap_extend_model

ID	13	
Title	xsap_extend_model	
Description	Extend model	
Parameters:		
	asynchronousExecution	[true, false]
	behaviourModel	
	faultExtensions	
	faultModes	
	timeout	

xSap Compute Fault Tree

O **OSLC Automation Plan**

14: xsap_compute_fault_tree

ID	14	
Title	xsap_compute_fault_tree	
Description	Fault tree generation	
Parameters:		
	algorithmType	[bmc, bdd, msat]
	asynchronousExecution	[true, false]
	behaviourModel	
	boundLength	
	computeProbability	
	faultModes	
	properties	
	propertyName	
	timeout	

xSap Compute FMEA Table

O OSLC Automation Plan

15: xsap_compute_fmea_table

ID	**15**	
Title	**xsap_compute_fmea_table**	
Description	**FMAE table generation**	
Parameters:		
	algorithmType	**[bmc, bdd, msat]**
	asynchronousExecution	**[true, false]**
	behaviourModel	
	boundLength	
	faultModes	
	properties	
	timeout	