OCORA

Open CCS On-board Reference Architecture

Tooling

Document ID: OCORA-BWS08-020

Version: 3.00

Release: R1

Date: 26.11.2021

Revision history

Version	Change Description	Initial	Date of change
1.00	▪ 1st Gamma version based on Beta incl. comments	RM	16-11-2020
1.10	▪ final for Gamma Release	RM	04-12-2020
1.20	▪ 1st Draft Delta version based on Gamma incl. comments	RM/TM	21-05-2021
1.30	▪ DRAFT version for Review	RM	17-06-2021
2.00	Official version for OCORA Delta Release	RM	30-06-2021
2.99	▪ DRAFT version for OCORA Release R1 Review	RM	16-11-2021
3.00	Official version for OCORA Release R1	RM	26-11-2021

Table of contents

Table of figures

Table of tables

References

Reader's note: please be aware that the numbers in square brackets, e.g. [1], as per the list of referenced documents below, is used throughout this document to indicate the references.

[1] OCORA-BWS01-010 – Release Notes
[2] OCORA-BWS01-020 – Glossary
[3] OCORA-BWS01-030 – Question and Answers
[4] OCORA-BWS01-040 – Feedback Form
[5] OCORA-BWS03-010 – Introduction to OCORA
[6] OCORA-BWS04-011 – Problem Statements
[7] OCORA-BWS08-010 – Methodology
[8] OCORA-TWS05-010 – Requirements – Management Guideline

Wherever a reference to a TSI-CCS SUBSET is used, the SUBSET is referenced directly (e.g. SUBSET-026). We always reference to the latest available official version of the SUBSET, unless indicated differently.

1 Introduction

1.1 Purpose of the document

The purpose of this document is to provide the OCORA participants with an overview of the tooling. It is and will remain a high level document with the aim to ensure common approach and basis and also give the freedom of choice in all not defined areas to the workstream leaders to define appropriate processes, methodology and tools if required for the specific type of work foreseen. Workstream leaders have the duty to report back to the methodology and tooling workstream any activities in such subjects and support alignment with already existing processes, methodology and tooling.

This document is addressed to experts in the CCS domain and to any other person, interested in the OCORA concepts for on-board CCS. The reader will gain insights regarding the topics listed in the table of content, and is invited to provide feedback to the OCORA collaboration and can, therefore, engage in shaping OCORA. Feedback to this document and to any other OCORA documentation can be given by using the feedback form [4].

If you are a railway undertaking, you may find useful information to compile tenders for OCORA compliant CCS building blocks, for tendering complete CCS system, or also for CCS replacements for functional upgrades or for life-cycle reasons.

If you are an organization interested in developing CCS building blocks according to the OCORA standard, information provided in this document can be used as input for your development.

1.2 Applicability of the document

This document is applicable for all OCORA workstreams. The OCORA Methodology [7] provides more insights in how the toolchain interacts.

OCORA tooling will be continuously revisited in order to serve the operating OCORA workstreams and proof sector compatibility with other ongoing sector initiative (e.g. RCA, ER JU, LinX4Rail).

1.3 Context of the document

This document is published as part of the OCORA release R1, together with the documents listed in the release notes [1]. It is the third release of this document which will be further developed in consecutive releases.

Before reading this document, it is recommended to read the Release Notes [1] and The OCORA Methodology [7]. If you are interested in the context and the motivation that drives OCORA we recommend to read the Introduction to OCORA [5], and the Problem Statements [6]. The reader should also be aware of the Glossary [2] and the Question and Answers [3].

2 Overview Tools

The OCORA collaboration require a wide set of tooling among project management, engineering, development, testing, etc. The evaluated toolchain focuses on the current project phase and related activities.

Tool Name	Purpose
MsOffice (Word, Excel, PowerPoint, Project)	General Purpose
MsTeams	Regular Exchange
Public Repository	*https://github.com/OCORA-Public*
Internal Repository	*https://github.com/openETCS/OCORA*
Polarion	Requirement Engineering and Management
Capella	Model-based Systems Engineering
Capella-Polarion integration using Eclipse plugin	Exporting Capella Modelling artefacts to Polarion
SCADE	Model-based Software Development

Table 1 - Overview Tools

Additional tools (ex. Use-Cases, Testing) might be listed at a later stage once the OCORA collaboration reaches later phases.

3 Internal Repository: GitHub

The OCORA internal repository can be found under: *https://github.com/openETCS/OCORA*

After each user is registered individually in GitHub. The OCORA Repository Administrator (*baseliyos.jacob@deutschebahn.com* or *rolf.muehlemann2@sbb.ch*) can grant access once the username is provided. GitHub as platform offers several additional functions (e.g., kanban boards are used in various workstreams and as overall workstream progress reporting, other like the software repository might be used at a later stage.

The use of the GitHub platform offers the necessary IT data backup process, as required by EN 50126-1 for quality management process. In addition, the OCORA Repository Administrator holds a local, manual copy of the repository including all data.

4 Public Repository: GitHub

The OCORA public repository can be found under: *https://github.com/OCORA-Public*
A user registration is only required in case of commenting.
The use of the GitHub platform offers the necessary IT data backup process, as required by EN 50126-1 for quality management process. In addition, the OCORA Repository Administrator holds a local, manual copy of the repository including all data.

5 Requirement Engineering and Management: Polarion

Polarion is a requirement management tool certified to allow safety relevant developments. It ensures required functionality such as collaboration, traceability and workflow to pass validation. Since Polarion is pre-certified for safety relevant developments and includes with ReqIF a possibility enabling lossless requirements exchange it is widely used in safety relevant requirement engineering and management.

The Requirements Management Guideline [8] provides the related information about the retention requirements, required by EN 50126-1 for quality management process.

6 Model-based Systems Engineering: Capella

Capella is an opensource software package for MBSE, supporting the Arcadia method. Hosted at polarsys.org, this solution provides a process and tooling for graphical modelling of systems, hardware or software architectures, in accordance with the principles and recommendations defined by the Arcadia method. Capella is mainly used for modelling complex and safety-critical systems in embedded systems development for industries such as aerospace, avionics, transportation, space, communications and security and automotive. For the following reasons, OCORA has decided to use Capella for MBSE:

- Capella is a dedicated, powerful tool to support the Arcadia method
- Most founding members of OCORA are using Capella in their CCS projects already

6.1 Capella/Polarion integration

OCORA has adopted Polarion as its Application Life-Cycle Management Tool covering quality assurance and requirements management. In addition, Polarion is also used for reviewing, approving, versioning and releasing/publishing of Capella artefacts.

To achieve that, Capella artefacts must be exported into Polarion. At the time of this writing, no official Capella-Polarion connector is available that would satisfy our needs. To fill the gap, an SBB custom-made Eclipse Plugin is used to export Capella artefacts to Polarion. The plugin is being developed and maintained by SBB. Having been used for over a year, it has reached a good level of maturity supporting both, the standalone Capella version, and the Teams for Capella version.

To successfully export Capella artefacts to Polarion, dedicated work item types must be configured in Polarion. That is why a dedicated Polarion project is used to collect the Capella exported artefacts (instead of linking it to the OCORA project).

The mapping of Capella artefacts to Polarion work items is managed by a customizable plugin configuration file. The base configuration exports the following artefacts:

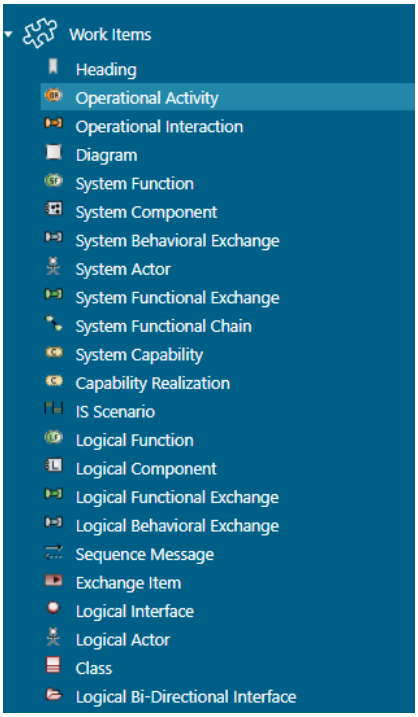

Figure 1 – Default Polarion Work Items for MBSE projects

In addition to the mapping of Capella artefacts to Polarion work item types, the configuration file also defines the Polarion project to connect with (see blue line in **Figure 2**).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- project:  "Capella", "SR40_Programm"  oder "Polarion_Schulung" -->
<polarionSyncConfig polarionUrl="https://trace.sbb.ch/polarion/ws/services/" project="OCORA_MBSE_Sandbox" polarionWorkItemTypeForDiagram="diagram">

    <!--   ++++++++++++++++++++++++   Operational Analysis   ++++++++++++++++++++++++   -->

    <typeMapping capellaName="OperationalActivity" polarionWorkItemType="operationalActivity"/>

    <typeMapping capellaName="FunctionalExchange" filterExpression="aql:self.ancestors()->filter(oa::OperationalAnalysis)->notEmpty()" polarionWorkItemType="operation
        <linkMapping capellaName="source" considerAncestors="true" polarionLinkRole="sourceOperational"/>
        <linkMapping capellaName="target" considerAncestors="true" polarionLinkRole="targetOperational"/>
    </typeMapping>

    <!--   ++++++++++++++++++++++++     System Analysis      ++++++++++++++++++++++++   -->

    <typeMapping capellaName="SystemFunction" polarionWorkItemType="systemFunction">
        <linkMapping capellaName="allocatorSystems" polarionLinkRole="isPerformedBy"/>
        <linkMapping capellaName="allocatorActors"  polarionLinkRole="isPerformedBy"/>
        <linkMapping capellaName="ownedFunctions"   polarionLinkRole="subfunction"/>
        <linkMapping capellaName="realizedOperationalActivities"   polarionLinkRole="realizes"/>
    </typeMapping>

    <!--
         The class "FunctionalExchange" has the feature "sourceFunctionOutputPort" and "targetFunctionInputPort", which points
         to the FunctionOutputPort/FunctionInputPort the FunctionPorts are contained in the function (are children), that means
         the function is the container (parent) of the FunctionPorts
      -->
    <typeMapping capellaName="FunctionalExchange" filterExpression="aql:self.ancestors()->filter(ctx::SystemAnalysis)->notEmpty()" polarionWorkItemType="systemFunctio
        <linkMapping capellaName="aql:self.sourceFunctionOutputPort.eContainer()" considerAncestors="true" polarionLinkRole="sourceSystem"/>
        <linkMapping capellaName="aql:self.targetFunctionInputPort.eContainer()"  considerAncestors="true" polarionLinkRole="targetSystem"/>
    </typeMapping>

    <typeMapping capellaName="Capability" filterExpression="aql:self.ancestors()->filter(ctx::SystemAnalysis)->notEmpty()"  polarionWorkItemType="systemCapability">
        <linkMapping capellaName="includedAbstractCapabilities"  polarionLinkRole="includes"/>
        <linkMapping capellaName="extendedAbstractCapabilities"  polarionLinkRole="extends"/>
    </typeMapping>

    <typeMapping capellaName="FunctionalChain" filterExpression="aql:self.ancestors()->filter(ctx::SystemAnalysis)->notEmpty()"  polarionWorkItemType="systemFunctiona
        <linkMapping capellaName="aql:self.eContainer()"                                                                                    polarionLinkRole="describes"/>
        <linkMapping capellaName="aql:self.ownedFunctionalChainInvolvements->filter(fa::FunctionalChainInvolvementFunction).involved" polarionLinkRole="involves"/>
        <linkMapping capellaName="aql:self.ownedFunctionalChainInvolvements->filter(fa::FunctionalChainInvolvementLink).involved"     polarionLinkRole="involves"/>
        <linkMapping capellaName="aql:self.ownedFunctionalChainInvolvements->filter(fa::FunctionalChainReference).involved"           polarionLinkRole="involves"/>
        <linkMapping capellaName="aql:self.involvingCapabilities"                                                                           polarionLinkRole="describes"/>
    </typeMapping>
```

Figure 2 – Example polarionconfig.xml

Being an Eclipse plugin, it is installed locally for each user. It provides a one-way only export of Capella artefacts to Polarion work items. Consequently, Capella always acts as the master of all artefacts. Polarion is only used to provide the necessary traceability for reviewing, approving, versioning and releasing/publishing artefacts. All changes required based on review comments must be applied in the Capella model.

It must be noted that the plugin doesn't perform any periodic/automatic exports. All exports to Polarion must be triggered manually by the user. This can be done for individual model elements or a selection of multiple model elements.

The plugin ensures that if a corresponding Polarion work item already exists for a Capella element, the work item is simply updated, otherwise a new work item is created. The link between Polarion work item and Capella artefact is identified using the Polarion ID and stored as an Extension Property of the corresponding Capella element. For Capella elements that do not support Extension properties (like diagrams) the plugin stores the Polarion ID as part of the element name, e.g. "[LAB] Localization Components {OMT-3070}".

A Capella element can only be re-exported to (updated in) Polarion if the corresponding work item has the state *Draft.* Otherwise the export will fail.

Currently there is neither an automated way of verifying that all Capella elements have been exported to Polarion nor a way of checking if all work items are up to date. A manual workflow/procedure is required to ensure this.

7 Model-based Software Development: Scade

ANSYS SCADE Suite is a model-based development environment for critical embedded software. With native integration of the formally defined SCADE language, SCADE Suite is the integrated design environment for critical applications including requirements management, model-based design, simulation, verification, qualifiable/certified code generation and interoperability with other development tools and platforms.

SCADE Suite is used to design critical software, such as rail interlocking systems and signalling, automatic train operation, computer-based train control, emergency braking systems, overspeed protection, train vacancy detection and many other aerospace, railway, energy, automotive and industrial applications.

So far OCORA has only used Model-based Software Development with Scade within the specific prototyping. For upcoming prototyping OCORA may or may not use Scade.