OCORA

Open CCS On-board Reference Architecture

PoC Configuration Management
Concept

Document ID: OCORA-TWS15-060

Version: 1.0

Date: 15.11.2023

Management Summary

This concept describes the approach and the state of TWS15: Prototyping Configuration Management Concept is described. Within this workstream a proof of concept of the OCORA Configuration Management Concept in the form of a prototype is carried out.

In the first, currently ongoing phase, the underlying Configuration Management Concept discussion paper is reviewed by the industry partner. This finalised concept, as well as the test cases to be carried out to prove its suitability, will subsequently be used by the industry partner to develop the necessary specifications for the prototype to be implemented.

Following the construction of the prototype hardware and the implementation of the necessary software functions, tests are carried out to prove the feasibility of the OCORA Configuration Management Concept from the proof of concept are documented and taken into account in the Configuration Management Concept.

Revision history

Version	Change Description	Initial	Date of change
0.1	▪ Initial draft	FT	07.06.2023
1.0	▪ Update for R5	FT	15.11.2023

Table of contents

Table of figures

Table of tables

Empty.

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

[1] OCORA-BWS01-010 – Release Notes

[2] OCORA-BWS01-040 – Feedback Form

[3] OCORA-TWS07-060 – OCORA Configuration Management – Concept

[4] Specification PoC SBB Maintainability, Selectron Systems AG, version 1.0, 15.11.2023 – available in the annex of the document

1 Introduction

1.1 Purpose of the document

The purpose of this document is to describe the proof of concept of the OCORA Configuration Management Concept.

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 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 [2].

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

If you are an organization interested in developing on-board 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

The document is currently considered informative but may become a standard at a later stage for OCORA compliant on-board CCS solutions. Subsequent releases of this document will be developed based on a modular and iterative approach, evolving within the progress of the OCORA collaboration.

1.3 Context of the document

This document is published as part of the OCORA Release 4, together with the documents listed in the release notes [1]. This document describes the concept and the way forward to gain the expected experience in Workstream 15 of the OCORA project.

2 Objectives of TWS15: Prototyping Configuration Management

Within the framework of the OCORA project, a concept [3] for configuration management was developed and published in the previous release within the Technical Workstream TWS07 Modular Safety, CENELEC, RAM. The concept describes how updates to the software of Building Blocks of the CCS subsystem and their configuration are prepared, initiated, executed and validated.

The goal of the Technical Workstream TWS15: Prototyping Configuration Management is to implement and validate the OCORA Configuration Management Concept by setting up a demonstrator. Gains in knowledge and experience are to be incorporated into the concept of TWS07.

The demonstrator is being carried out together with an experienced company from the railway industry supplying hardware and software components for safety-relevant functions on rail vehicles. The Lyss (CH)-based company Selectron Systems AG could be contractedfor its implementation.

3 Proceeding

3.1 Review of OCORA Configuration Management Concept

The first step of the TWS is a review of the existing version of OCORA Configuration Management Concept by Selectron Systems staff.

Up to now, remote updates on SW parts of rail vehicles have only been implemented to a very limited extent in the railway sector. The workstream is not aware of any examples of over-the-air software updates, particularly for functionalities that are relevant to safety.

Therefore, the aim of the review is to compare the view and procedures of the OCORA concept with the knowledge and experience of the industry and to adapt the concept to best practices as far as possible without negative influence on the functionality. This would simplify later consideration of the concept on a broad scale.

The discussion in detail concerns the following content:

Review and discussion of the process management under the following conditions

- Delineation of the concept's horizon of consideration
- Ensuring remote updates from different HW hierarchy levels
- Ensuring remote updates for functionally dependent Building Blocks
- Analysis and adaptation of the procedure to ensure the highest possible availability".

Definition of activation conditions for the distribution and installation of an update

- for the Configuration Manager
- for the Building Blocks

Definition of the exported integration, operation and deployment constraints

- Narrowing down the applicability of the concept
- Definition of integration conditions
- Monitoring, checks and tests by personnel
- Necessity of manual procedures and processes in case of failure

Assignment of resulting responsibilities per role

As part of OCORA Release R5, an update of the Configuration Management Concept is published based on the review and the associated discussions held with Selectron.

The concept will be professionally reviewed by an ISA before finalisation. The review is limited to the identification of safety and security deficiencies and gaps in the safety and security concept. Within the proof of concept no certification is sought, but the certifiability should also be considered.

3.2 Requirement specification for the demonstrator

In the following, joint specification of the procedure, elements and activation conditions for distribution, authorization and activation of safe and non-safe app updates (and rollbacks) for the demonstrator shall be elaborated. Basically, the specification are based on the revised OCORA Configuration Management Concept. However, a complete implementation would be very extensive and time-consuming. Therefore, the specification shall reduce the extent in a matter, that the objectives of the proof of concept are still fulfilled. To increase efficiency, the demonstrator should enable the integration of existing software components of Selectron Systems.

The specification is to be developed by Selectron Systems. The OCORA team's task is to carry out regular reviews and ensure the project objectives are met.

A key element is the definition of the activation conditions of the updates, i.e. compliance with the necessary conditions for update notification, software download, as well as the installation of the update.

The specification has been finalised in various sesseions and is attached to this document as an appendix [4].

3.3 Developing and setting up objectives of the demonstrator and the test cases

A set of specifications drawn up by SBB detailing the objectives is to serve as the basis for the development of the test cases. The elaboration of the contents is still ongoing. An excerpt of the list of objectives is shown below.

Validation of all possibilities of the statuses of the flow chart of the OCORA concept for configuration management

Carrying out the updates for the listed cases:

- Update of software of a Building Block
- Update of software of a building block, which is functional unidirectionally linked with another Building Block
- Update of software of a building block, which is functional bidirectionally linked with another Building Block

Simplified creation and compilation of the update files according to the necessity of the update case and roles

Simplified dispatching the update to the Configuration Manager

Checking of the update files by the Configuration Manager on:

- Correctness
- Completeness

Transmission of the software update to Building Block according to activation conditions

Checking of the software update files by the Building Block on:

- Correctness
- Completeness

Execution of the software update according to activation conditions

Consideration of error management

Status feedback from the Building Block to the Configuration Manager

Status feedback of the Configuration Manager to the update server

3.4 Implementation of the demonstrator and execution of the test runs

The demonstrator will then be implemented based on the developed and reviewed specification. A simplification of the demonstrator's structure is shown in Figure 1.

Two demonstrators will be implemented, whereas one will be located at SBB and the other at Selectron Systems. Applications with and without safety-related functions are implemented on each of these. A training for the SBB employees by Selectron Systems should enable SBB to carry out the test runs independently.

The findings from the testing will be edited in a concept validation report and incorporated into the next OCORA release. If necessary, the OCORA concept for configuration management will be updated based on the outcome of the prototyping.

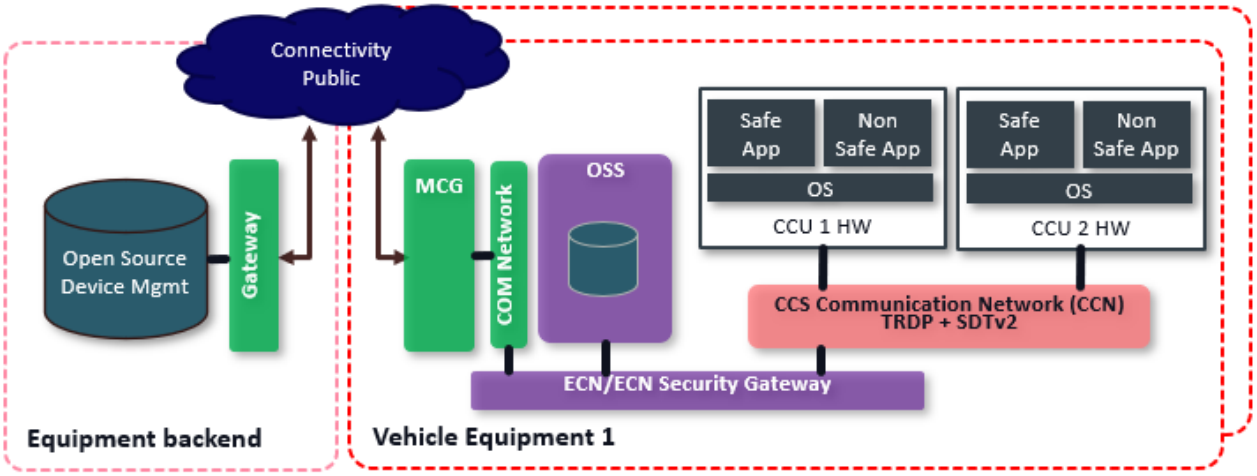


Figure 1: Schematic diagram of the demonstrator from the tender agreement with Selectron Systems AG

4 Annex

Document *Specification PoC SBB Maintainability, Selectron Systems AG, version 1.0, 15.11.2023*

Specification
PoC SBB Maintainability

Document identification:	OCORA SRS PoC Maintainability_v1.0.docx
Authors:	Christian Dudka, Gerold Flaskaemper
Version:	1.0
Date:	15.11.2023
Release:	-

Content

Table of Figures

1. Document Identification

1.1 History

Output No.	Name	Date	Change in chapter	Reason for change
0.1	Christian Dudka	04.09.2023	All	Initial version
0.2	Christian Dudka	12.09.2023	All	Internal review
0.3	Christian Dudka	19.09.2023 20.09.2023	All 6	1st review with SBB Manifest requirements added
0.4	Christian Dudka	21.09.2023	2, (3), 4, 5, 6	2nd review with SBB, reworked, (only wording changed), CM state chart added, CM requirement for update sequence added
0.5	Christian Dudka Gerold Flaskaemper	09.10.2023	1 – 6 7	Internal review with infoteam Chapter implementation added
0.6	Christian Dudka Gerold Flaskaemper	24.10.2023	1-5 6	3rd review with SBB, findings corrected Only JSON examples corrected
0.7	Christian Dudka	30.10.2023	4 – 6	4th review with SBB, findings corrected, introducing REST for the CM/BB-UM interface, manual and auto mode of CM described, changed wording of interactions between BB-UMs and CM to consider client-server architecture, normal and extended logging described. Example for Vehicle JSON-Configuration refined.
0.8	Christian Dudka	08.11.2023	6	5th review with SBB, findings for Manifest specifications corrected, new discussed Config Types and Elements added and described. JSON-examples updated.
1.0	Christian Dudka	15.11.2023	6	6th review with SBB, usage of references in JSON-example refined. Specification (chapter 1 – 6) released for PoC.

1.2 Purpose and Scope

The purpose and scope of this PoC specification shall be to provide and demonstrate a simple, realizable approach of the SBB Configuration Management concept [R3] of publishing and applying components of a *Vehicle Configuration* to a small On-Board system consisting of a *Configuration Manager* running on a Raspberry Pi and two CPUs.

1.3 References

No.	Name	Identification
[R1]	Abbreviations and Definitions	GI5100-01
[R2]	OCORA Glossary	OCORA-BWS01-020_Glossary.pdf
[R3]	OCORA Configuration Management Concept	OCORA-TWS07-060_Configuration-Management-Concept_V2.docx
[R4]	Explanation of REST*ful* API	restfulapi.net
[R5]		
[R6]		
[R7]		
[R8]		
[R9]		
[R10]		

1.4 Abbreviations and definitions

Refer to document "Abbreviations and Definitions" [R1] and [R2]. The following important abbreviations are listed redundantly for easy reference.

BB	Building Block
BB-UM	Building Block Update Manager
BIL	Basic Integrity Level
CCS	Control-Command and Signaling
CCS-OB	CCS On Board
CLI	Command Line Interface
CM	Configuration Manager
DAS	Driver Advisory System (BIL)
ETP	European Train Protection (SIL4)
ETCS	European Train Control System
IEC	International Electrotechnical Commission (referring to IEC 1131 – standard for process control software)
OCORA	Open CCS On-board Reference Architecture
OB	On-Board
PRAMSS	Performance, Reliability, Availability, Maintainability, Safety and Security
PTU-OS	Physical Train Unit Operation System
REST API	REST stands for representational state transfer [R4]. A REST API (also known as REST*ful* API) is an application programming interface that conforms to the constraints of REST architectural style and allows interactions with REST*ful* web services. The constraints of REST includes • Client-server architecture with requests managed through HTTP • Stateless client-server communication • Cacheable data that streamlines client-server interactions • Uniform interface between components • A layered system that organizes each type of server • Code-on-demand (optional)
SSH	Secure Shell network protocol
SIL	Safety Integrity Level
TCMS	Train Control and Management System
VC	Vehicle Configuration

1.4.1 Definition of a Manifest

A Manifest shall be a complete, clear, precise and approved human readable description of a Vehicle, System or Building Block Configuration using a standardized lightweight data-interchange format. A Manifest shall include references to dependent components and subordinate Manifests and shall contain elements for verification of identification, authentication and compatibility.

1.4.2 Definition of a Configuration

A Configuration shall be a description of all items (components) required to operate a physical instance of a Vehicle, System or Building Block. It shall include default values for parameters.

1.4.3 Definition of a Building Block

A Building Block is an entity, having standardized functionality, standardized PRAMSS requirements (including Tolerable Functional Failure Rate [TFFR], Safety Integrity Level [SIL] and Safety Related Application Conditions [SRAC]), standardized interfaces (on all OSI Layers) towards other Building Blocks and/or external systems. Building Blocks are exchangeable and migratable, without impacting other Building Blocks. Building Blocks may be purchased from different suppliers and capable of being integrated by a third party.

1.4.4 Definition of a Package

A Package shall consist of its Manifest and of all items (components) referenced by its Manifest. If other Manifests are referenced, the derived items shall be included as well. The integrity respectively folder and file structure specified by the Manifests shall be strictly respected. A package shall be creatable analogous to its Manifest(s) on each level: Vehicle, System and Building Block.

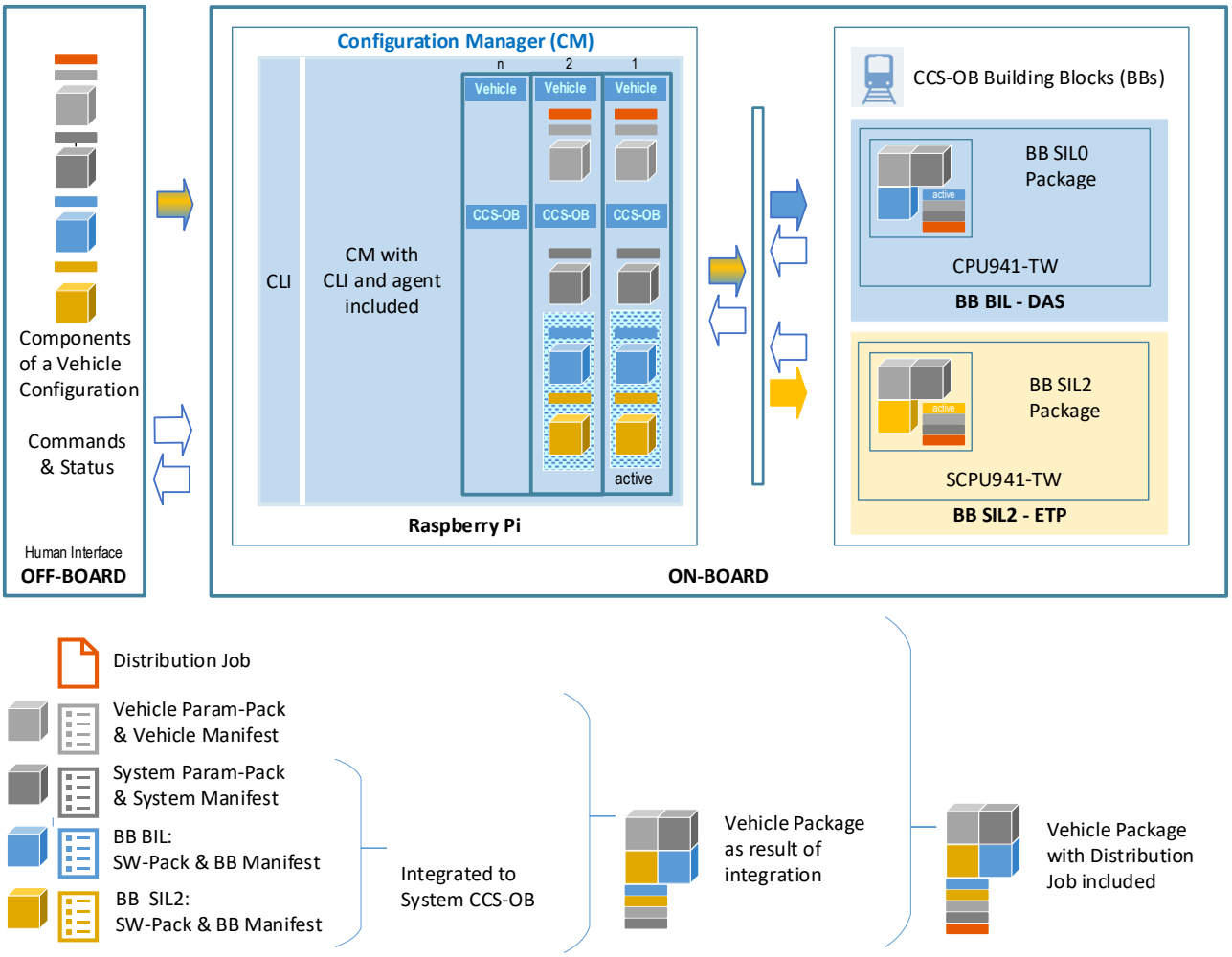
2. Introduction and Overview

As part of the OCORA project, SBB has developed a high-level Configuration Management concept for railway equipment. Goal of this specification is to demonstrate the proposed update process using modified Selectron components.

The OCORA Configuration and Management Concept [R3] of SBB shall be the base for this PoC maintainability specification. The scope of this PoC specification, however, shall be to provide and demonstrate a realizable approach of publishing and applying components of a *Vehicle Configuration* to the On-Board system.

The concept, [R3] figure 3, is simplified to serve the essential requirements for the PoC with focus on the On-Board side and without the sophisticated Off-Board aspects. The following figure provides an overview and illustrates the application of the components of a Vehicle Configuration to the On-Board system consisting of the Configuration Manager and just two Building Blocks (BBs).

Figure 1: Overview of the On-Board system with CM and BBs



A Vehicle Configuration shall contain its Manifest and a Param-Pack. For the PoC the Distribution Job is not considered and replaced by an activation command. The Param-Pack includes data necessary for the integration process. All referenced components in the Manifest shall belong to

the Configuration. These shall be basically the subordinate Building Blocks (BBs) each with its own subordinate Manifest and SW-Pack. A SW-Pack contains all necessary signed firmware parts and configurations, that are correctly parameterized for a successful installation and operation on a BB. Optionally the BBs can be grouped to logical entities e.g. BB BIL DAS and BB SIL2 ERP to System CCS-OB. See chapter 1.4 for definitions of Manifest, Configuration and Package.

For the PoC the Off-Board system shall be simplified and reduced to an operator sending commands to and retrieving status information from the On-Board system via command line (CLI). The On-Board system shall consist of the Configuration Manager (CM) and the Building Blocks (BBs). The agent and repository according to [R3] figure 3 shall be integrated to the CM. The BBs shall be just represented by two CPUs: one CPU941-TW dealing with a BB BIL e.g. for DAS and another SCPU941-TW dealing with a BB SIL2 e.g. for ETP.

The Vehicle Configuration with its referenced components shall be organized as filesystem with a folder file structure on the Configuration Manager (CM). The components of a Vehicle Configuration shall be exchangeable with a file transfer protocol. Different Vehicle Configurations shall be storable on the CM. On top level of the Vehicle Configuration the Vehicle folder shall be located and on bottom level the folder(s) for the Building Block(s). It shall be optional to group the BBs to logical coherent entities in intermediate folders.

The CM shall wait until it is requested by a BB to provide the correct BB Manifests of the Vehicle Package to the CPUs: In case of Normal Operation the BB Manifest with BIL integrity to the CPU941-TW and the BB Manifest with SIL2 integrity to the SCPU941-TW. Thereby the result of every interaction shall be sent to the CM and stored in a logfile. The content of the logfile shall be readable by command via CLI.

In the following chapters the use cases for the PoC and requirements for the CM, BBs and the Manifest are described in detail.

3. Required scenarios and use cases

In the interaction of Manifests, Configuration Manager (CM) and Building Blocks (BB) the PoC shall include the following scenarios “Normal Operation, Update, Rollback and Error Handling” demonstrated by the corresponding use cases. The basic idea shall be that the CM is the master of the update process of the BBs and has complete control over it. The CM shall be able to request for the actual status of the BBs in order to re-initiate synchronization between them.

3.1 Execute Normal Operation

3.1.1 Normal Operation on startup

On startup after switching to the parent state Operational Mode each BB shall request for the actual BB Manifest. The received BB Manifest shall be verified and compared with its own BB Manifest. If the CM is not available, the verification and comparison shall be skipped. Normal Operation shall be proceeded, if the stored BB Package and BB Manifest on the CPU are valid. If a BB Manifest from the CM was received, the BB informs the CM about its status of conformity with the active configuration. The actual Vehicle Configuration with status of all referenced BB participants including life checks shall be stored and available to be queried.

3.1.2 Normal Operation in progress

In Normal Operation the BB shall keep requesting the CM periodically for new BB Manifests. The received BB Manifest shall be verified and compared with its own BB Manifest. If the CM is not available, the verification and comparison shall be skipped. If a BB Manifest from the CM was received, the BB shall inform the CM about its status of conformity with the actual configuration. Thereby the CM assures that all BBs are periodically checking for their current Manifest (life check).

3.1.3 The CM provides compliance of update package

After startup and after every periodical request for BB Manifests the CM shall gather the outcome of all participating BBs and provide information about compliance of the active Vehicle Configuration including hierarchical structure, files, versions, and feedback from the BBs (status of conformity with the actual configuration on the CPUs).

3.2 Update

The process of applying a new BB Configuration, started by activating of a new Vehicle Configuration, is described in Figure 2.

3.2.1 Transfer components of a new Vehicle Configuration to the CM

Via file transfer new components of a Vehicle Configuration shall be stored for the CM. The CM shall verify by command the syntax, semantic and integrity of the components with the help of the included Vehicle Manifest and shall provide the result of the verification. The activation of a specific Vehicle Configuration shall be done by command. The activation shall be a replacement of the Distribution Job.

3.2.2 Activate new Vehicle Configuration

Once the new components referenced by the Vehicle Configuration have been activated, the CM shall respond with the correct new BB Manifest to all requests of the BBs. The BB verifies compliance and authentication of its new BB Manifest and informs the CM about its status of conformity with the new configuration including its recommendation for an update.

3.2.3 Start update process

The CM gathers the status of all BBs in a particular group and decides whether an update can be started. If an update is required, the CM requests all BBs to stop operational activities and proceed to state Update Pending. Each BB checks with the help of the IEC Application, if it is safe to stop functional operation and proceed with the update. If all conditions are met, the BB accepts the request and stops all operations. In any case the CM is informed whether the request is accepted or declined. The CM gathers the status of all BBs, repeats declined requests and waits for all to accept the update request.

Once all BBs are ready to start the update process, the CM starts the update by sending an update trigger to all BBs.

3.2.4 Apply Update

Immediately after the BB has received the update trigger and turned from state Update Pending to state Update, it requests the BB Package addressed by the new BB Manifest from the CM. Once the download of the BB Package is completed, the BB verifies its integrity, identity and authentication. On success the new BB Package with its SW-Packs including firmware and parameters are installed. The BB informs the CM about its status of conformity with the actual installation and turns in any case to state Update Completed. The progress of the update of each participating BB and the overall progress of all participants according to the Vehicle Configuration shall be provided.

3.2.5 Resume Operation

Once the BB changed to state Update Completed, it shall wait for further instructions from the CM: either changing to Normal Operation, initiating a system shutdown/ restart or an eventual rollback. In case of a restart the BB shall resume its previous update complete state and inform the CM about its actual status.

3.3 Rollback

3.3.1 Rollback triggered by the CM

The rollback possibility is part of the update process and allows the system to switch back to the previous BB Package in case the installation of a new BB Package has failed. Therefore, in the state Update Completed the BB shall inform the CM about the update result. If the installation of a new BB Package has failed, the CM shall offer the possibility to force either a re-update of the new BB Package or to force a rollback to the previous BB Package. The rollback is done by setting the previous components of the Vehicle Package as the active one and by retriggering the update process.

The use case of a rollback of an “old” device from stock, where the original Configuration is unknown by the CM, is not part of the PoC.

3.4 Error handling

3.4.1 List of possible errors

The following errors shall be handled:

- Missing BB Manifest
- Conformity of BB Manifest failed
- Integrity of BB Manifest failed
- Identity of BB Package failed
- Authentication of BB Package failed
- Installation of new BB Package failed
- Rollback failed

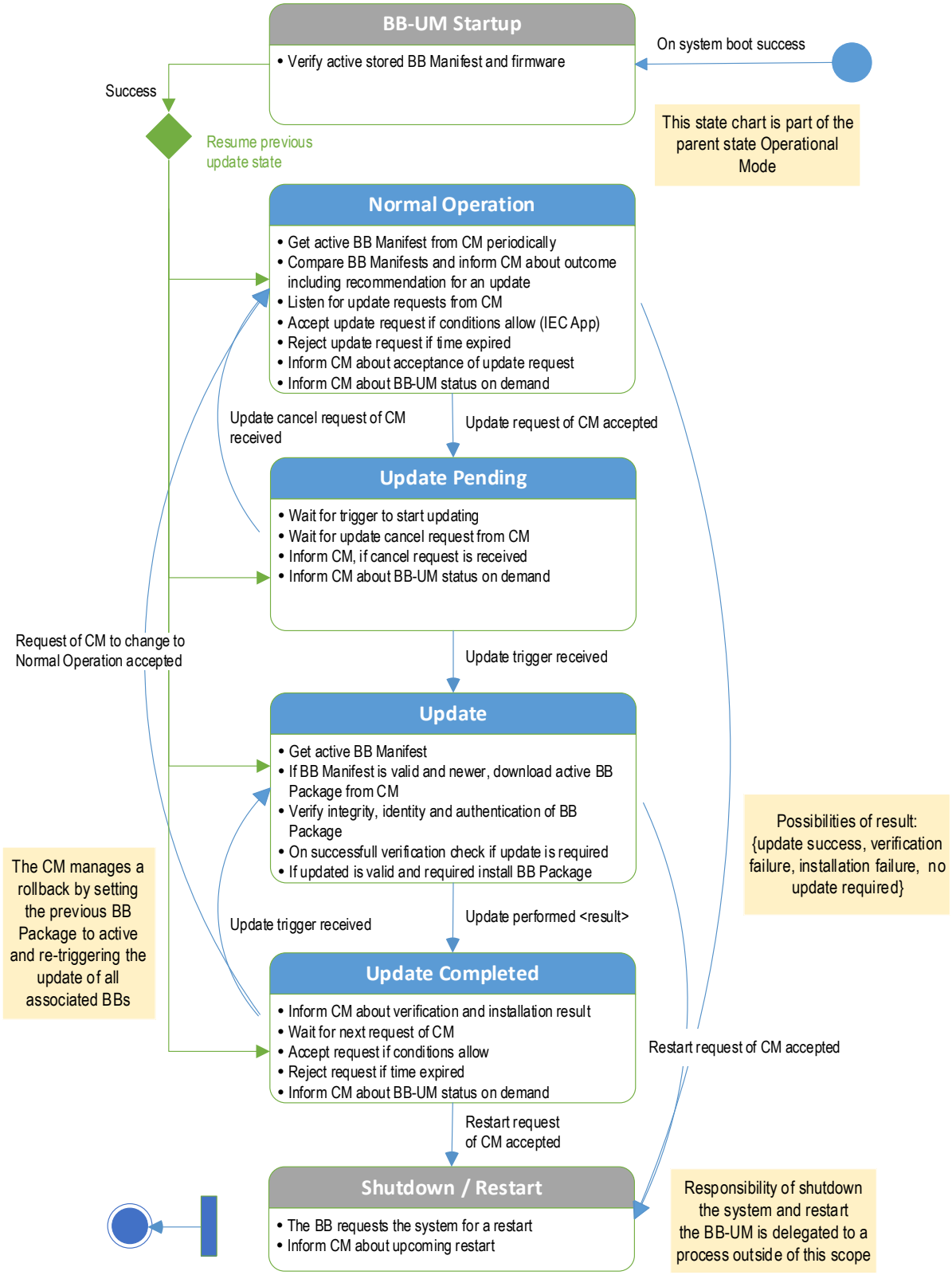
3.4.2 The CM is not available on startup

If the CM is not available on startup and the BB has already stored a valid and authentic BB Configuration, Normal Operation shall proceed and synchronization of its BB Manifest with the CM is done later on when the next periodical check is executed.

4. Building Block requirement specification

Figure 2: State chart of the BB Update Manager Operational Mode

The state chart of the BB-UM Operational Mode illustrates update process and interaction with the Configuration Manager (CM).



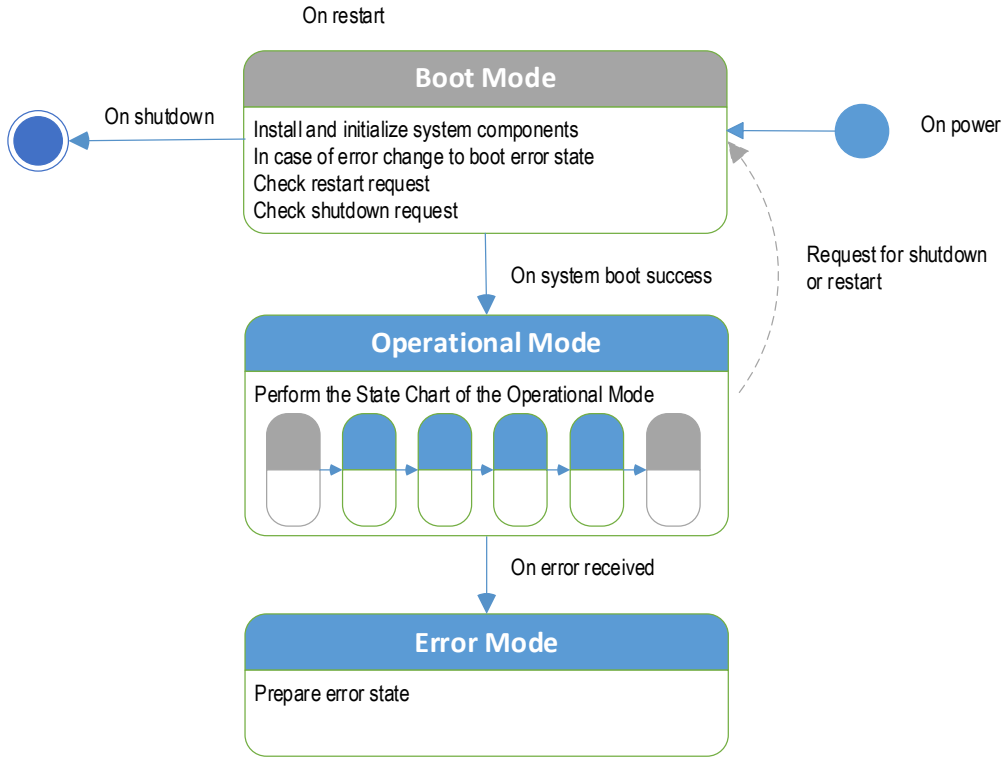
The state chart of the BB-UM above illustrates the four update states Normal Operation, Update Pending, Update and Update Completed. Moreover, there is a startup entry and a shutdown/ restart exit state. The sequence is part of the parent state Operational. Another parent state shall be Error. Since the CM shall be able to request for a restart, the Update states shall be memorized and resumed on the subsequent startup. Therefore, it shall be in the responsibility of the CM to request for most state transitions. Startup and shutdown (program termination) is in the responsibility of a process outside the illustrated scope.

The verification of the BB Package on startup includes checking the BB Manifest for identity and authenticity and executing the signature verification of the firmware.

Whenever after a CM request the BB response depends on a condition that must be fulfilled in the overall system (IEC application), the BB shall check the response time from the overall system to be able to reject the CM request after a defined time interval.

Figure 3: State chart of the BB Update Manager parent states

The state chart of the BB-UM parent states illustrates the interaction of the states Boot Mode, Operational Mode, and Error Mode.



The Boot Mode is not part of the BB-UM but responsible for executing shutdown or restart requests. On installation success the BB-UM shall be started in Operational Mode. In case of an error the BB-UM changes to Error Mode where it shall remain until the system receives a restart or shutdown request.

4.1 Interfaces of the BB Update Manager

The firmware part of a BB that is responsible for the update process e. g. in the CPU shall be named BB Update Manager (BB-UM).

4.1.1 Configuration Manager interface

As the BB-UM is the counterpart of the CM it reflects almost all requirements. The communication between the CM and its participating BB-UMs shall use a REST*ful* architecture style approach based on the HTTP protocol [R4].

4.1.1.1 BB-UM get its BB Manifest

The BB-UM shall get on startup and later on in state Normal Operation periodically its BB Manifest from the CM. If the CM is not available, the BB-UM shall proceed without an error.

4.1.1.2 BB-UM responds the outcome of its BB Manifest comparison

After a BB Manifest was downloaded from the CM, the BB-UM shall compare the received BB Manifest with its current configuration (e.g. software versions, configuration hashes). The verification result, syntax, integrity, identity, authenticity and recommendation for an update shall be responded to the CM as status message.

4.1.1.3 BB-UM receives update request

Depending on the outcome and recommendation for an update the CM shall request the BB-UM to perform an update. The BB-UM therefore shall receive the request and verify the conditions for acceptance. The update permission shall be requested from the IEC Application with safety level integrity. Once the acceptance is available, the BB-UM shall change from state Normal Operation to state Update Pending and inform the CM that the update request was accepted. If an acceptance of the update request is not available within a certain time, the BB-UM shall send an update rejected to the CM. It shall be in the responsibility of the CM to retry the request.

4.1.1.4 BB-UM waits for update trigger in state Update Pending

After the BB-UM has accepted an update request from the CM, it waits in the state Update Pending for an update trigger. The trigger shall be sent by the CM when the conditions are fulfilled.

4.1.1.5 BB-UM starts update

After the BB-UM has received an update trigger, the BB-UM shall change from state Update Pending to state Update and download initially the active BB Manifest *). After the BB Manifest has been downloaded, the BB-UM verifies the update condition. If the downloaded BB Manifest doesn’t match with its installed configuration the BB-UM shall download its BB Package (including SW-Pack with firmware and parameters). The gathered BB Package shall be verified by the BB-UM referring to syntax, integrity, identity and authenticity.

*) In this state the BB-UM has already downloaded an active BB Manifest from the CM. The initial repetition of downloading the BB Manifest is necessary, because a transition to state Update shall be possible not only from state Update Pending but as well from state Update Completed. In case of a rollback the CM has changed the active BB Manifest in the meantime.

4.1.1.6 BB-UM responds with outcome of the update

If the verification of the BB Package was successful, the BB Package shall be installed. In case of successful installation or failure the BB-UM shall change from state Update to state Update Completed and send the outcome of the update as status message to the CM. It shall be in the responsibility of the CM to react accordingly and send a corresponding state transition command to the BB-UM.

Possible outcomes of the update shall be: update successful, verification failure, installation failure, no update required.

4.1.1.7 BB-UM receives change to Normal Operation

Once an update is successfully completed, the BB-UM shall accept a change from state Update Completed to state Normal Operation from the CM, if the conditions allow this transition. The BB-UM shall inform the CM via status message about the acceptance of the request.

4.1.1.8 BB-UM receives an update trigger in state Update Completed

Once an update is finished and the BB-UM has entered state Update Completed, the BB-UM shall accept an update trigger from the CM, if the conditions allow this transition. The BB-UM shall inform the CM via status message about the acceptance of the request. If the request is accepted, the BB-UM shall change from state Update Completed to state Update again.

It shall be controlled by the CM, if a rollback to the previous configuration or a retrial of the present configuration shall be executed. The CM shall therefore set the corresponding Vehicle Configuration to active before it sends the start update trigger.

4.1.1.9 BB-UM receives a restart request

The BB Update Manager shall accept a restart request from the CM in the states Normal Operation, Update and Update Completed, if the conditions allow this transition. The reset permission shall be requested from the IEC Application with safety integrity. The BB Update Manager shall inform the CM via status message about the acceptance of the request before the request is delegated to the system.

4.1.1.10 BB-UM receives update cancel request

Once the update request was sent to the BB Update Manager, the BB shall wait for the start update trigger. In the state Update Pending the BB Update Manager therefore shall accept a cancel request from the CM and change to state Normal Operation. The BB Update Manager shall inform the CM via status message about the acceptance of the request.

4.1.2 IEC App interface

For the PoC the number of vPLCs on the xCPU941-TW shall be reduced to just one vPLC. The IEC Application is running in a safety integrity part of the system and shall therefore be queried for safety relevant permissions.

4.1.2.1 The BB-UM gets status from IEC App run/stop

The BB-UM shall have the possibility to get the status from the IEC App. If the application is stopped, an update request from the CM forwarded from the BB-UM to the IEC App cannot be answered. In this case the BB-UM shall send an update rejected to the CM. It is up to the CM to retry the update request.

4.1.2.2 The BB-UM gets permission from IEC App for update

After the BB-UM receives an update request from the CM, it shall request the IEC App for update permission. The IEC App shall signal the permission to the BB-UM if the required conditions are fulfilled. For the PoC a simple approach shall be considered. If an acceptance of the update request is not available within a certain time, the BB-UM shall send an update rejected to the CM. It shall be in the responsibility of the CM to retry the request.

4.1.2.3 The BB-UM gets permission from IEC App for shutdown/restart

After the BB-UM receives shutdown/ restart request from the CM, it shall request the IEC App for shutdown/restart permission. The IEC App shall signal the permission to the BB-UM if the required conditions are fulfilled. For the PoC a simple approach shall be considered. If an acceptance of the shutdown/ restart request is not available within a certain time, the BB-UM shall send a restart rejected to the CM. It shall be in the responsibility of the CM to retry the request.

4.1.2.4 The BB-UM sends SW versions to the IEC App

The BB-UM verifies its stored BB Manifest and BB Package and shall send the versions and status of the components to the IEC App.

4.2 Common requirements of the BB

4.2.1 BB-UM receives turn to error state request

Upon fatal error of the system the BB-UM shall accept a turn to error state request. If an exception in the system is detected that forces the safe state of the device, the entry process into the safe state of the system shall also inform the BB-UM to turn to its safe error state.

5. Configuration Manager requirement specification

Figure 4: State chart of the CM

The state chart of the CM illustrates the interaction with the CLI and the state machine of the BB-UM.

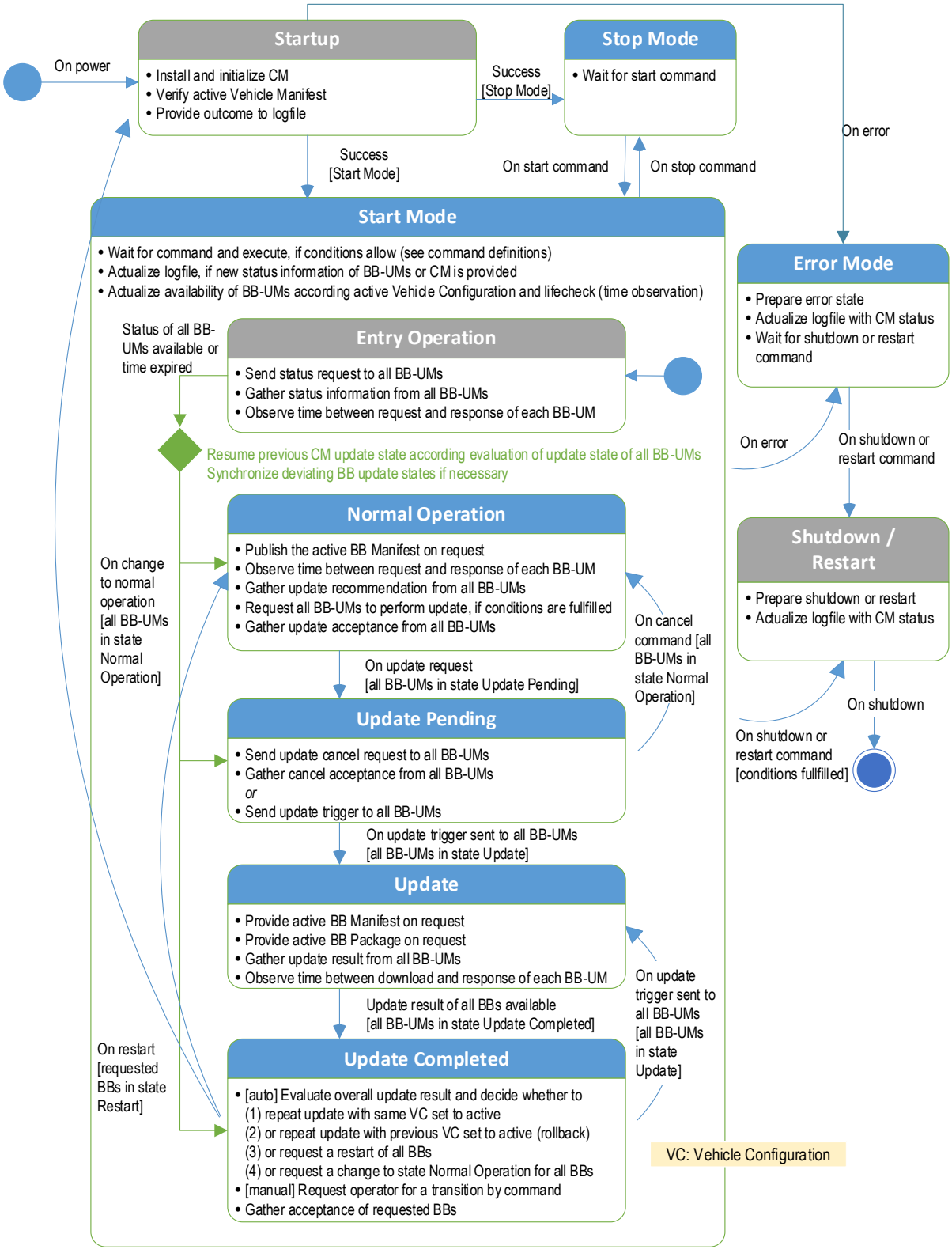
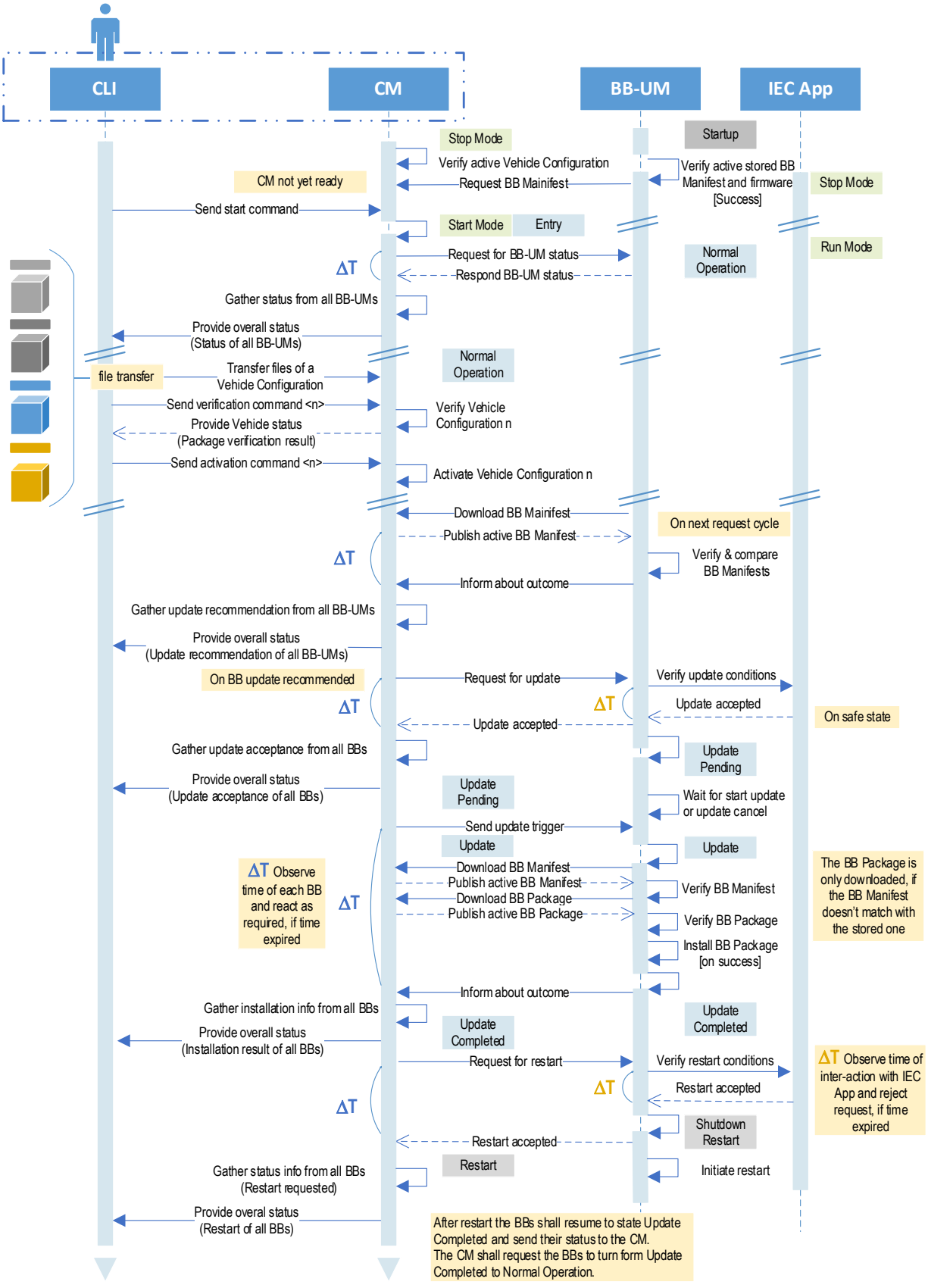


Figure 5: Sequence of applying new components of a Vehicle Configuration to the CM



The sequence above illustrates the context of the different participants while activating a new vehicle configuration of the CM according to the described use cases. Only the success sequence without error handling is considered.

5.1 Interfaces of the Configuration Manager

For the Configuration Manager (CM) the following interfaces shall be present:

5.1.1 File transfer to the CM

5.1.1.1 CM file transfer protocol

Via SSH a new Vehicle Configuration with its folders, files and items shall be transferable to the CM.

5.1.1.2 CM storage capacity

The CM shall have at least a capacity to store ten Vehicle Configuration with all its folders, files and items.

5.1.2 CLI command line interface

The CLI is part of the agent – for the PoC both included in the CM. The protocol for exchanging data between a human operator and the CM shall be SSH.

5.1.2.1 Start/ Stop the activity of the CM

The activity of the CM shall be controllable by a stop respectively a start command. The start/ stop state shall be stored persistently and resumed on startup. If set to stop, the CM shall not request or respond to the connected BBs. On set to start the CM shall synchronize with the BBs automatically by requesting for their status.

5.1.2.2 Set active Vehicle Configuration

Since the CM shall store a few Vehicle Configurations, it shall be possible two set the active configuration by command via CLI. The index of the activation shall be stored persistently and shall be resumed on restart.

5.1.2.3 Verify active Vehicle Configuration on startup

On startup the active Vehicle Configuration shall be verified referring syntax, integrity, identity and authenticity.

5.1.2.4 Provide status of the Vehicle Manifest

The status of a stored Vehicle Configurations of the CM shall be evaluated and queried by command. Since each Vehicle Configuration includes referenced BB Manifests, the status summarize the verification check of all depending Manifests referring to integrity, identity and authenticity. Moreover the status shall list name, version and the verification check of each dependent Manifest separately. The referenced firmware and parameter packages of each Manifest shall also be listed including names, versions and verification checks of the referenced files.

5.1.2.5 Provide overall status of the participating BBs

Since a started CM gathers status information from the connected BBs and stores it into a logfile, the overall status shall be provided by command. See BB status message definition.

5.1.2.6 Provide commands to trigger BB state transitions manually

Via CLI it shall be possible to send commands to the CM that are forwarded to the BBs to influence the state transitions of the BB-UMs manually. See Command list definitions.

5.1.2.7 Auto and manual mode of the CM

The CM shall be set by command either to auto or to manual mode. The two modes shall have an effect on how the different state transitions of the CM shall be executed:

Event	Auto mode	Manual mode
On startup of the CM:	The CM shall be started automatically.	The CM shall be started by a start command.
On all associated BB-UMs have provided their update state after startup:	If not all BB-UMs are in the same state, the CM synchronizes the states of the BB-UMs automatically by either changing all to state Normal Operation or state Update.	If not all associated BB-UMs are in the same state, the states of the BB-UMs are listed and the user is prompted to synchronize the states by command.
On all associated BB-UMs have provided their update recommendation:	The CM shall request all associated BB-UMs for an update automatically.	The CM shall prompt to send an update request. The update request is sent by an update request command.
On all associated BB-UMs have accepted an update request:	The CM shall trigger an update of all associated BBs automatically.	The CM shall prompt to send an update trigger or update cancel. The update trigger is sent by an update trigger command.
On one or more associated BB-UMs have rejected an update request:	The CM shall request all associated BB-UMs for an update again automatically. If the repetition was not successfully, the CM shall prompt for a decision by user interaction.	The CM shall prompt for a decision by user interaction.
On all associated BB-UMs have completed their update:	The CM shall evaluate the overall update result of all associated BB-UMs and decides, whether to trigger an update (retrial or rollback *)), restart the system or change to state Normal Operation automatically. *) Rollback shall be done by setting previous VC to active in advance.	The CM shall list the update result and prompt for an user interaction, either to repeat the update, execute a rollback *), restart the system or change to state Normal Operation. *) Rollback shall be done by setting previous VC to active in advance.
On exception in auto mode:	If the internal logic of the CM is not in a position to decide the next step or transition automatically, it shall prompt for a decision by user interaction.	none
On fatal error of the CM:	The CM shall prompt for a shutdown or restart command.	The CM shall prompt for a shutdown or restart command.

5.1.3 BB-UM interface

As the BB-UMs are the counterpart of the CM, they reflect almost all interface definitions. The communication between the CM and its participating BB-UMs shall use a REST*ful* architecture style approach based on the HTTP protocol [R4].

5.1.3.1 CM gathers current BB-UM status

After the CM is started and changed to Entry Operation it shall gather the status of all BB-UMs by request and store a summary of the outcome that can be queried by command.

5.1.3.2 CM synchronizes status of the BB-UMs

The CM shall evaluate the outcome of the status of all BB-UMs on Entry Operation and decide to which CM state it shall enter. In case not all BB-UMs have the same state, the CM shall evaluate, if a synchronization of the BB-UM states is required.

5.1.3.3 CM evaluates availability of all BB-UMs

The CM shall extract the participating BBs of the vehicle out of the active Vehicle Manifest. On every request to the BB-UMs in every state the CM shall observe the time between the request respectively and the responds for each associated BB-UM to verify their availability and proper operation. In case the specified time has expired, the CM shall set the life check status of the BBs accordingly.

5.1.3.4 CM provides BB Manifest

On Normal Operation a BB-UM shall download its BB Manifest from the CM periodically and compare it with its stored BB Manifest. The CM shall therefore provide the active BB Manifest to the BB-UM.

5.1.3.5 CM gathers update recommendation

After BB Manifest comparison, the BB-UM shall send the actual update recommendation to the CM. The CM shall therefore gather the update status from all participating BB-UMs and store a summary of the outcome that can be queried by command.

5.1.3.6 CM sends update request

If all participating BB-UMs have sent their update recommendation and at least an update to one BB is necessary, the CM shall sent an update request to all BB-UMs. The BB-UMs decide itself if an update shall be executed. When the conditions are fulfilled, the BB-UM shall respond with update accepted respectively on time expiration update rejected.

5.1.3.7 CM gathers update accepted

After all BB-UMs have received their update request, the CM shall gather the update accepted status from the BB-UMs and provide a summary of the outcome. In case of a rejection of one or more BB-UMs the CM shall retry the update request or prompt for user interaction*). If all BB-UMs have sent their acceptance respectively turned to state Update Pending the CM shall change to state Update Pending as well.

*) Look up 5.1.2.7 manual or auto mode

5.1.3.8 CM evaluates practicable update sequence

The CM shall be in a position to optimize the update process and determine a logical and practicable update sequence automatically out of the BB dependency description of the BB Manifests. If no BB dependency description is defined, the CM shall serve the BB-UMs in the order of the incoming BB-UM update requests.

5.1.3.9 CM sends update trigger

If all concerned BB-UMs are in state Update Pending, the CM shall send an update trigger to all BB-UMs to start the update. When all BB-UMs have changed to state Update the CM shall change to state Update as well.

5.1.3.10 CM provides BB Package

After a BB-UM has received its update trigger and therefore changed to state Update, it immediately downloads the active BB Manifest and verifies, if the BB Manifest matches with the actual stored BB Manifest *). Only if the BB Manifest doesn't match the BB-UM shall download its active BB Package including firmware and parameters from the CM.

*) In this state the BB-UM has already downloaded an active BB Manifest from the CM. The initial repetition of downloading the BB Manifest is necessary, because a transition to state Update shall be possible not only from state Update Pending but as well from state Update Completed. In case of a rollback the CM has changed the active BB Manifest in the meantime.

5.1.3.11 CM gathers update result

After all concerned BB-UMs have downloaded their BB Package, the CM shall wait and gather the result of the update and installation from the BB-UMs and provide a summary of the outcome. The outcome of the installation shall be differentiated between update successful, verification failure, installation failure, no update required. If all associated BB-UMs have sent their update result to the CM, the CM shall turn to state Update Completed as well.

5.1.3.12 CM requests to restart the system

Once an update and installation are completed successfully the CM shall have the possibility to request the BB-UMs to change to state Restart to request a restart of the system. The BB-UMs shall respond with restart accepted before the restart is executed.

5.1.3.13 CM requests change to Normal Operation

Once an update and installation are completed successfully and a restart was already executed the BB-UMs shall resume to state Update Completed. The CM shall have the possibility to request the BB-UMs to change to state Normal Operation. The BB-UMs shall respond with "change to Normal Operation" accepted. If all associated BB-UMs have sent their acceptance to change to state Normal Operation, the CM shall change to state Normal Operation as well.

5.1.3.14 CM requests re-update or rollback

If the outcome of the latest update process indicates a failure on one or more BB-UMs, the CM shall have the possibility to send an update trigger to the associated BB-UMs again. Compare 5.1.3.9. In case of a rollback **) the CM shall set the previous BB Manifest to active in advance, before the update of all associated BB-UMs is triggered again.

**) The use case of a rollback of an "old" device from stock, where the original Configuration is unknown by the CM, is not part of the PoC.

5.1.3.15 CM request to cancel update

Once the update request was sent from the CM to all associated BB-UMs, the BB-UMs are waiting for the trigger from the IEC-App indicating update is permitted. The CM therefore shall have the possibility to cancel the pending update and allow the BB-UMs to return from state Update Pending to state Normal Operation.

5.1.3.16 CM changes to error state

Upon fatal error of the CM, the CM shall change to state Error and prompt for a shutdown or restart command.

5.2 Common requirements of the CM

5.2.1 Status message definitions

A status message sent from a BB-UM to the CM shall contain the following information:

- Status of the active BB Manifest on the BB-UM: name, version, verification result referring integrity, identifier and authenticity
- Update recommendation status <yes, no>
- Update accept status <yes, no>
- Installation result <undefined, update success, verification failure, installation failure, no update required>
- Update State of the BB-UM

5.2.2 Command definitions

The CLI shall offer operational commands for the CM and test commands forwarded via CM to the participating BB-UMs to take effect on the state transitions of the BB-UMs manually.

5.2.2.1 Operational commands

- Operational command *start* Configuration Manager
- Operational command *stop* Configuration Manager
- Operational command *vcs* all available Vehicle Configurations are listed
- Operational command *activate vc <1..n>* only Vehicle Manifest n is activated (all others are set to deactivated)
- Operational command *verify vc <1..n>* verify and list status of Vehicle Configuration n *)
- Operational command *list bb <1..n>* list status information of BB-UM with index n
- Operational command *mode <auto, manual>* set mode of CM to auto or manual
- Operational command *version* the name and version of the CM is displayed
- Operational command *log <normal, extended>* the logging mode of the CM is set

*) Recursive for all files dependent from and referenced by the Vehicle Manifest)

5.2.2.2 Test commands

- Test command *request update* request update of all associated BBs. Restricted to state Normal Operation
- Test command *cancel update* cancel update of all associated BBs. Restricted to state Update Pending to cancel an already

- Test command *trigger update* trigger update of all associated BBs. Restricted to states Update Pending and Update Completed to trigger an already accepted update respectively repeat an update in case of failure
- Test command *restart* <all, n> restricted to state Update Completed
- Test command *normal operation* <all, n> restricted to state Update Completed
- Test command *accept update* <all, n> for the PoC to simulate the verification of the update condition executed normally from the IEC App.

5.2.3 Logging of status information

The CM shall gather and update the status information (see 5.2.1) including life check of all BB-UMs and write it into a logfile. The content of the logfile shall be queriable by different commands via CLI (see 5.2.2) or event driven displayed to the CLI.

Event	Normal mode	Extended mode
On startup of the CM:	The CM shall prompt name, state and version “CM is operating in start/stop mode vx.x.x”.	The CM shall prompt name, state and version “CM is operating in start/stop mode vx.x.x”.
On status information is requested from the CM:	-	The CM shall display the action.
On initial status information of a BB-UM is available:	-	The CM shall display the status result of the BB-UM.
On providing BB Manifests to the BB-UMs:	-	The CM shall display the action initially of each BB-UM with name and identifier that starts the download.
On download recommendation is available of all associated BB-UMs:	The CM shall display a summary of the download recommendation.	The CM shall display a summary of the download recommendation and additionally list the download recommendation of each associated BB-UM.
On update is requested from the CM:	-	The CM shall display the action.
On update acceptance is available of all associated BB-UMs:	The CM shall display a summary of the update acceptance.	The CM shall display a summary of the update acceptance and additionally list the acceptance of each associated BB-UM.
On update trigger is sent to all BB-UMs:	-	The CM shall display the action.
On update cancel is sent to all BB-UMs:	-	The CM shall display the action.
On providing BB Package to the BB-UMs:	-	The CM shall display the action initially of each BB-UM with name and identifier that starts the download.
On update result is available of all associated BB-UMs:	The CM shall display a summary of the update result.	The CM shall display a summary of the update result and additionally list the result of each associated BB-UM.
On restart request of the CM is sent to all associated BB-UMs:	The CM shall display the action.	The CM shall display the action.
On change to Normal Operation is sent to all associated BB-UMs:	The CM shall display the action.	The CM shall display the action.
On error request is received:	The CM shall display the action.	The CM shall display the action.
On exception in auto mode is occurred:	The CM shall display the action.	The CM shall display the action.

6. Manifest requirement specification

The following chapters describe the definition and requirements of a Manifest and its depending items. The goal is the formulation of a generic approach on the vehicle, system and device level. This approach thus deviates from the original specification [R3] in chapter 5.

6.1 Requirements

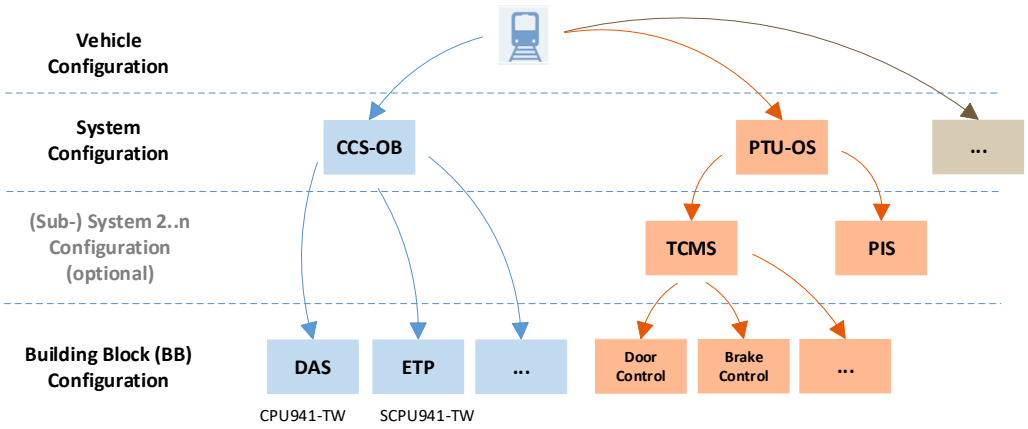
6.1.1 File format of a Manifest

Since the Manifest shall be written in a human readable, standardized lightweight data-interchange format, the Manifest shall be written preferably in the JSON file format.

6.1.2 Pattern of a Manifest

Since each level and instance shall have its own Configuration respectively Manifest, the individual Manifests in the same file must be combinable analogous to a hierarchal tree pattern or referenceable in case of distribution to different Manifest files. The granularity of the breakdown into different Manifest files shall be freely selectable with the goal of being able to realize an overall structure that is as clear and as readable as possible. The optional grouping into system and sub-systems is therefore only a structural aid and has no further function. The Manifest for the vehicle, system and Building Blocks (device) are mandatory – the optional grouping into one or more system levels is optional and depends on the complexity and logical respectively physical relations of the whole system.

Figure 6: Hierarchical levels of the different Configurations / Manifests



6.1.3 Mandatory and optional elements of the Manifest

To cover diverse aspects on the different configuration levels with a generic approach, mandatory and optional elements shall be configurable.

6.1.4 Support of referenced files

Since different Manifests shall be chainable and also referenceable in case of distribution to different Manifest files, handling of the different files shall be supported. Moreover within the

Manifest structure it shall also be possible to reference to other files like software and parameter packages.

6.1.5 Integrity

The integrity of the Manifest with all its referenced and nested Manifests and items shall be checked by the parser.

6.1.6 Identification and authentication

With the elements of the Manifest it shall be possible to identify the target of a file and verify its authenticity (source supplier, integrator, distributor). To support the identification and authentication process, all entities involved in the Configuration Management shall use unique identifiers. Cryptographic techniques shall therefore be applied. In conclusion, a Configuration shall only be activated after a positive verification of the authenticity of its source, ascertainment that the source is a trusted Configuration Management entity, and that the configuration is targeted to the correct receiving entity.

6.1.7 Compatibility and versions of different BBs

To support the integrator in the process of building the different Manifests, a functional version x.y.z shall be used to validate the compatibility of the Building Blocks to be integrated. The functional version shall consist of a major (x), minor (y) and a patch (z) index. If BB_1 depends on BB_2, they shall be compatible, if their major indices are equal $x_1 = x_2$ and their minor indices fulfill the equation $y_2 \geq y_1$. The patch indices z_1 and z_2 can take any integer $z \varepsilon N_0$ and have no relevance for the compatibility verification.

6.1.8 BB dependencies and update hierarchy

The Building Blocks can have dependencies to other Building Blocks. The definition of such dependencies shall be represented in the BB Manifest. The defined dependencies shall contain the required version. The CM shall therefore be able to optimize the update process and determine a logical and practicable update sequence automatically. The dependency definitions shall be mandatory.

6.1.9 Parser capabilities

The parser for the Manifests and referenced items shall fulfill the following requirements.

6.1.9.1 Reading information

The parser shall read and extract the corresponding information of each item out of the Manifest with respect to the used file format.

6.1.9.2 Retrieving file paths

Since a Manifest references other Manifests the parser shall retrieve the corresponding file paths and names and continue parsing within the referenced Manifest according to the nested Manifest structure.

6.1.9.3 Checking syntax and semantic

The parser shall check the syntax and semantics of the Manifests with respect to all nesting levels. In case of an error the parser shall give a helpful hint to correct the syntax or semantic of the Manifests accordingly.

6.1.9.4 Checking integrity

The parser shall check the integrity of all referenced Manifests and referenced items with respect to all nesting levels. In case of an error the parser shall give a helpful hint to correct the missing or incomplete part of the Manifests accordingly.

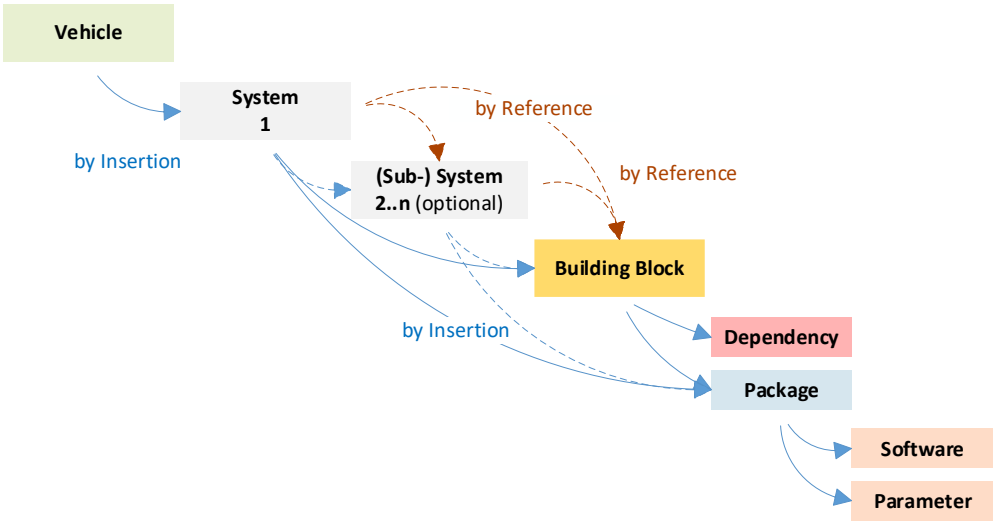
6.1.10 Config Items and Config Types in the Manifest structure

A Manifest consists of Config Items associated to different Config Types. The following two figures illustrate the dependency respectively the nesting possibilities of different Config Types.

6.1.10.1 Dependency of Config Types

The dotted lines indicate optional or alternative dependencies. If the Manifests are spitted in different files they shall be included by reference. The Vehicle Manifest contains Items of type System, Building Block and Package. The BB Manifests contains Items of type Dependency and Package. The Items of type Package reference to Items of type Software or Parameter.

Figure 7: Dependency of Config Types



6.1.10.2 Nesting possibility of Config Types

Package Items shall be allowed to be integrated in the Vehicle, System and Building Block Items. However, the Package Items for Vehicle and System Items shall only contain Parameter Items – Software Items shall only be allowed in combination with Building Block Items.

Figure 8: Nesting possibilities of Config Types

Config Type \ Config Item	Vehicle	System	Building Block	Depend-ency	Package	Software	Parameter
Vehicle	x	x	x	x	x	x	x
System	✓	✓	x	x	x	x	x
Building Block	x	✓	x	x	x	x	x
Dependency	x	x	✓	x	x	x	x
Package	x	✓	✓	x	x	x	x
Software	x	x	x	x	✓	x	x
Parameter	x	x	x	x	✓	x	x

6.1.11 Config Items and its Elements

Each Config Item has a set of Elements and the Element Config Type specifies the type of the Config Item. Some of the Elements shall be used for all Config Types and some of them are type specific. Some of them shall be mandatory (M), some of them are optional (O) and some of them unused (-) for the corresponding Config Item. If an Element has a default value, the explicit usage of the Element shall be optional.

Figure 9: Config Items and Elements of a Manifest

The following chart defines the usage of the Elements for each Config Item.

Elements \ Config Item	Vehicle	System 1 *mandatory*	(Sub-) System 2..n *optional*	Building Block	Depend-ency	Pack-age	Soft-ware	Param-eter
✓ **Config Type** *String specifies the type of a Config Item*	M	M	M	M	M	M	M	M
✓ **Type ID** *String with UUID format*	M	M	M	M	M	M	M	M
✓ **Config ID** *String with UUID format*	M	M	M	M	-	-	-	-
✓ **Name** *String*	O	O	O	O	O	O	O	O
✓ **Description** *String*	O	O	O	O	O	O	O	O
✓ **Supplier** *String*	O	O	O	O	O	O	O	O
✓ **Metadata** *String*	O	O	O	O	O	O	O	O
✓ **Functional Version** *String with x.y.z format*	-	-	-	M	M	-	-	-
✓ **Release Version** *String with x.y.z format*	M	M	M	M	-	M	M	M
✓ **Manifest** *String with relative path to a json-file*	-	-	-	-	-	-	-	-
✓ **Archive** *String with relative path to a tar.gz-file*	-	-	-	-	-	-	M	M
✓ **Network IDs** List of *Strings with IPv4 format*	-	O	O	M	-	-	-	-
✓ **Systems** *List of Items of type System*	M	O	O	-	-	-	-	-
✓ **Building Blocks** *List of Items of type BB*	-	M	M	-	-	-	-	-
✓ **Packages** *List of Items of type Software or Parameter*	-	O	O	M	-	-	-	-
✓ **Reference** *Triple with three Elements*	-	O	O	O	-	-	-	-
✓ **Dependencies** *List of Items of type Dependency*	-	-	-	M	-	-	-	-
✓ **Hash Type** *String SHA256 default*	-	-	-	-	-	-	O	O
✓ **Hash** *String coded according Hash Type*	-	-	-	-	-	-	M	M
✓ **Signature Type** *String PKCS#1 default*	O	O	O	O	-	-	-	-
✓ **Signature** *String coded according Signature Type*	M	M	M	M	-	-	-	-

6.1.12 Explanation of the Elements

6.1.12.1 Config Type

By introducing the Element Config Type, type-related differences regarding type specific, optional and mandatory Elements shall be checked in the Manifest. The format of this Element shall be a string and only defined Config Types shall be allowed by the parser. The Config Type shall be defined as an ENUM to specify the allowed values. Config Types are:

- Vehicle
- System
- Building Block
- Dependency
- Package
- Software
- Parameter

6.1.12.2 Type ID and Config ID

Each Config Type shall have an Element Type ID and each Manifest in a separate JSON file shall have an Element Config ID. For a Config or Type ID an Universally Unique Identifier (UUID) shall be used. The composition of the UUID is not yet specified but for a Building Block Configuration may be based on a combination of a datetime value and a part of the MAC address of the device – and for a Vehicle Configuration generated by hashing a combination of elements in the Manifest. For the Network Identifier a list of IP-addresses (IPv4) shall be used. The format of the identifiers shall be a string and the parser shall check the specific UUID respectively IP format.

6.1.12.3 Name, Description, Supplier and Metadata

Some additional Metadata shall be used to describe Manifest specific information. Essential metadata are Name, Description and Supplier. The format of these Elements shall be a string.

6.1.12.4 Release and Functional Version

The Element Release Version is provided from a supplier for a Building Block, Package, Firmware or Parameter Item or from an Integrator for a Vehicle or System Manifest. The Element Functional Version instead shall guarantee the compatibility between different Building Blocks of different suppliers. The type of the version shall be a *x.y.z* coded string (see 6.1.7).

6.1.12.5 Manifest

The Element Manifest is part of the Vehicle or System Config Item and shall reference to a dependent Manifest in a separate file with the extension *json*. The used path shall be relative and the format shall be a path coded string <*./path/name.json*>. The Element is optional because nested Manifests shall also be directly includable in their parent Vehicle or System Manifest.

6.1.12.6 Archive

The Element Archive is part of the Software or Parameter Config Item and shall reference to a separate file with the extension *zip* or *tar.gz* including firmware or parameter settings. For the PoC only *tar.gz* shall be supported. The used path shall be relative and the format shall be a path coded string <*./path/name.tar.gz*>.

6.1.12.7 Network IDs

The Element Network IDs shall be a list of strings that define the used and allowed IPv4 addresses on the local network. For each BB the definition shall be mandatory but it shall be also definable on level of the Vehicle or System Manifest. If no definitions are made on BB Manifest level the addresses shall be derived from the parent levels.

6.1.12.8 Systems

An Element Systems shall be a possibility to combine any number of Building Blocks to logical coherent sub-systems, e.g. BB “DAS” and BB “ERP” are grouped to system “CCS-OB” (see Figure 6). Only the initial System in the Vehicle Manifest is mandatory – further nesting is optional, e.g. the BB “Door Control” and BB “Brake Control” are grouped to sub-system “TCMS” (see Figure 6). The optional grouping is only a structural aid and has no further function. Nested Systems of any number shall be possible as composition or reference to another System Manifest.

6.1.12.9 Building Blocks

The Element Building Blocks shall be used within a Vehicle or System Manifest and include all associated BB Manifests either as reference to a file or directly inserted with its Config Items and Elements. Building Blocks shall contain only Dependency and Package Items and within the Package Items Software and Parameter Items shall be allowed.

6.1.12.10 Packages

The Element Packages shall be used within a Vehicle, System or BB Manifest and include all associated Software and Parameter Items. However Software Items are only allowed in Packages used by Building Blocks.

6.1.12.11 Reference

The Element Reference shall be used to refer to nested (Sub-)System- or BB-Manifests that are stored in separate files. The Element Reference shall be a Triple containing three Elements: Description, Signature and Manifest. While parsing a Manifest a Reference shall be completely replaced by the referenced Manifest.

6.1.12.12 Dependencies

The Element Dependencies shall be used to link dependent BBs together. The Type ID of the dependent BB and the Functional Version shall be assigned to each Dependency to allow automatically executed compatibility checks.

6.1.12.13 Hash Type

The default value for the Hash Type shall be SHA256. Other Hash Types are not supported for the PoC.

6.1.12.14 Hash

A Hash shall be assigned to each Archive referenced within the Software or Parameter Item of a Package. Hashing shall be done according to the Hash Type. It shall be used to verify the integrity and identity of the referenced Archive and to detect any tampering and modifications.

6.1.12.15 Signature Type

The default value for the Signature Type shall be PKCS#1. Other Signature Types are not supported for the PoC.

6.1.12.16 Signature

A cryptographic algorithm shall be used to sign the hashed content of every Manifest. If a BB Manifest is included within a Vehicle or System Manifest by composition (and not by reference) the corresponding Building Block shall additionally be secured by a signature. The signature shall be used for integrity-, identity- and authentication checks. This allows to detect any tampering, modifications and to verify the root of trust.

6.1.13 Example JSON files

The examples for a JSON file for the Vehicle Configuration with a System named CCS-OB and two Building Blocks DAS and ERP lean on the overview described in Figure 1 for the PoC. The System is composed within the Vehicle Manifest and not excluded to a separate file. For illustration purposes the BB DAS ist configured as a composition and the BB ETP as reference.

6.1.13.1 Example Manifest of a Vehicle Configuration


```
{
    "Config Type" : "Vehicle",
    "Config ID" : "8259d77b-e4e7-4261-8025-fa92c3a0a795",
    "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac120002",
    "Name" : "Track Runner 2",
    "Supplier" : "Acme Corp.",
    "Release Version" : "1.1.1",
    "Systems" : [
        {

            "Name" : "CCS-OB",
            "Config Type" : "System",
            "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac1102345",
            "Description" : "Control-Command and Signaling On Board",
            "Release Version" : "1.1.0",
            "Building Blocks" : [
                {
                    "Config Type" : "Building Block",
                    "Config ID" : "8259d77b-e4e7-4261-8025-fa92c3a0a795",
                    "Type ID" : "6fcf44f5-f9a5-4da1-93e9-8c27ffe02d36",
                    "Name" : "DAS",
                    "Release Version" : "1.0.0",
                    "Functional Version" : "2.1.0",
                    "Supplier" : "Octan",
                    "Network IDs" : [
                        "255.255.255.192",
                        "255.255.255.193"
                    ],
                    "Dependencies" : [
                        {
                            "Config Type" : "Dependency"
                            "Type ID" : "6e8fa7b0-7fe2-4d08-9101-b22f086f0787",
                            "Name" : "ETP",
                            "Functional Version" : "1.0.0"
                        }
                    ],
                    "Packages" : [
                        {
                            "Type ID" : "fce031f8-7462-4ad1-b320-7fcba3329baa",
                            "Config Type" : "Software",
                            "Name" : "Firmware xCPU94x Core",
                            "Release Version" : "1.1.2",
                            "Signature" : "81dc096b6ab943bfa6c03961fed1c8c72745cff5382ddec…",
                            "Archive" : "./xCPU94x/xCPU94x Core Firmware SELE V1.1.2.tar.gz"
                        },
                        {
                            "Type ID" : "e5c2c137-ee35-4044-a41b-42169e9d2b22",
                            "Config Type" : "Parameter",
                            "Name" : "Parameter xCPU94x Core",
                            "Release Version" : "1.1.0",
                            "Signature" : "fb6aefb7562f4817a865e5f2e9fcfbd99da2e1f15c774aec123…",
                            "Archive" : "./xCPU94x/xCPU94x_Core_Parameter_SELE_V1.1.0.tar.gz"
                        }
                    ]
                },
                {
                    "Reference" : {
                        "Description" : "ETP",
                        "Signature" : "6f049c1ec76f49ac82c1474084421abc4661be1978cfdab1c093…",
                        "Manifest" : "./System/CCS-OB/BBs/ETP_v1.1.9.json"
                    }
                }
```

```
            ],
            "Signature" : "07d385472a15456fb12687bf9ddbbfc153de81ba616a44b3b0195256cf5b5ac8"
        },
        {
            "Packages" : [
                {
                    "Config Type" : "Parameter",
                    "Type ID" : "6c4d65f4-2572-11ee-be56-0242ac120003",
                    "Description" : "Vehicle fe0001 Parameter",
                    "Release Version" : "1.1.9",
                    "Archive" : "./Vehicle_fe0001_Parameter_v1.1.9.zip",
                    "Signature" : "9785eb76c2ea023ca2f4904b9cb5672d62dc78b2acf28d702827376a2"
                }
            ]
        }
    ],
    "Signature" : "07d385472a15456fb12687bf9ddbbfc153de81ba6b3b0195256cf5b5ac8"
}
```


6.1.13.2 Example Manifest of a Building Block

The referenced BB ETP Manifest from the previous example:


```
{
    "Config Type" : "Building Block",
    "Config ID" : "8ba8d7a2-5f09-46bd-adf7-ccace4400d41",
    "Type ID" : "9f671cb3-c213-4c71-a15d-0e25fb0ccab1",
    "Name" : "ETP",
    "Description" : "This is the all new, highly performant ETP implementation",
    "Supplier" : "Hulesch & Quenzel",
    "Release Version" : "1.1.9",
    "Functional Version" : "1.1.0",
    "Network IDs" : [
            "255.255.255.192"
    ],
    "Dependencies" : [
        {
            "Type ID" : "6e8fa7b0-7fe2-4d08-9101-b22f086f0787",
            "Config Type" : "Dependency",
            "Name" : "DAS",
            "Functional Version" : "2.1.0"
        }
    ],
    "Packages" : [
        {
            "Type ID" : "fce031f8-7462-4ad1-b320-7fcba3329baa",
            "Config Type" : "Software",
            "Name" : "Firmware xCPU94x Core",
            "Release Version" : "1.1.2",
            "Signature" : "a22d3d8621ae4221be96e915cde4c1f2fc055ccf1c4f40b3a033e16057baddaf",
            "Archive" : "./xCPU94x/xCPU94x_Core_Firmware_SERP_V1.1.2.tar.gz"
        },
        {
            "Type ID" : "e5c2c137-ee35-4044-a41b-42169e9d2b22",
            "Config Type" : "Parameter",
            "Name" : "Parameter xCPU94x Core",
            "Release Version" : "1.1.0",
            "Signature" : "70e4fc8ed40742f4aabef9767f19a67f605689649b814123a78a0198e3dfa16b",
            "Archive" : "./xCPU94x/xCPU94x_Core_Parameter_SERP_V1.1.0.tar.gz"
        }
    ],
    "Signature" : "9cad6fc0cdca4f2cb5356563c1ebc1aef3836d1124de484bb606ce8e3e0887e3"
}
```

6.1.13.3 Example Vehicle Manifest with nested Systems

The following example illustrates a Vehicle Manifest using Sub-Systems according Figure 6. The example is uncomplete and not all mandatory Elements are listed to illustrate the nesting of Systems in a compact way:

```
{
    "Config Type" : "Vehicle",
    "Name" : "Track Runner 3",
    "Release Version" : "2.2.0",
    "Systems" : [
        {
            "Name" : "CCS-OB",
            "Config Type" : "System",
            "Description" : "Element 1 of System",
            "Building Blocks" : [
                {
                    "Config Type" : "Building Block",
                    "Name" : "DAS",
                    "Packages" : [
                        {
                            "Config Type" : "Software",
                            "Archive" : "./xCPU94x/xCPU94x_Core_Firmware_DAS_V1.1.2.tar.gz"
                        }
                    ]
                },
                {
                    "Config Type" : "Building Block",
                    "Name" : "ETP",
                    "Packages" : [
                        {
                            "Config Type" : "Software",
                            "Archive" : "./xCPU94x/xCPU94x_Core_Firmware_ETP_V1.1.2.tar.gz"
                        }
                    ]
                }
            ]
        },
        {
            "Name" : "PTU-OS",
            "Config Type" : "System",
            "Description" : "Element 2 of System with 2 Sub-Systems",
            "Systems" : [
                {
                    "Name" : "TCMS",
                    "Config Type" : "System",
                    "Description" : "Element 1 of Sub-System PTU-OS",
                    "Building Blocks" : [
                        {
                            "Config Type" : "Building Block",
                            "Name" : "Door Control",
                            "Packages" : [
                                {
                                    "Config Type" : "Software",
                                    "Archive" : "./xCPU94x/xCPU94x Core Firmware DCL V1.1.2.tar.gz"
                                }
                            ]
                        },
                        {
                            "Config Type" : "Building Block",
                            "Name" : "Brake Control",
                            "Packages" : [
                                {
                                    "Config Type" : "Software",
                                    "Archive" : "./xCPU94x/xCPU94x_Core_Firmware_BCL_V1.1.2.tar.gz"
                                }
                            ]
                        }
                    ]
                },
                {
                    "Name" : "PIS",
                    "Config Type" : "System",
                    "Description" : "Element 2 of Sub-System PTU-OS",
                    "Building Blocks" : [
                        {
```

```
                            "Config Type" : "Building Block"
                        }
                    ]
                }
            ]
        }
    ],
    "Signature" : "07d385472a15456fb12687bf9ddbbfc153de81ba616a44b3b0195256cf5b5ac8"
}
```