OETCS/WP3/D3.5.1.1

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

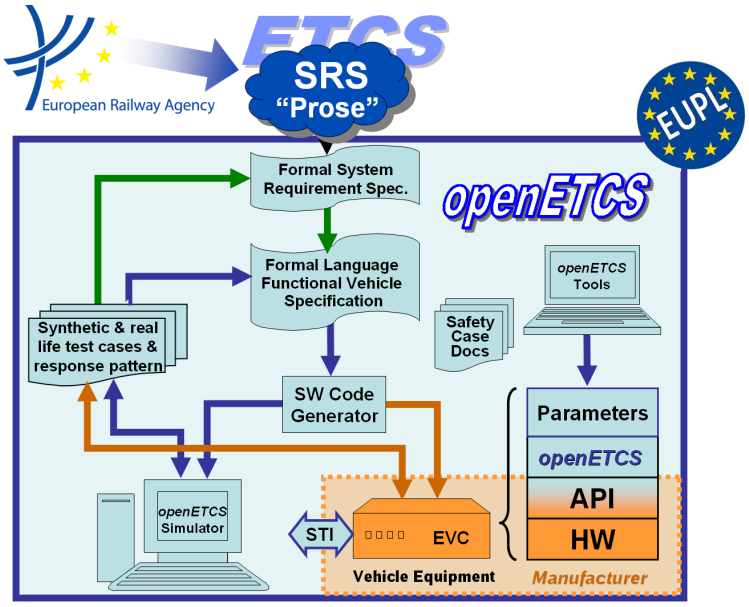
Work-Package 3: “Modeling”

openETCS System Architecture and Design Specification

First Iteration: ETCS Kernel Functions

Bernd Hekele, Peter Mahlmann, Peyman Farhangi, Uwe Steinke and Christian Stahl

September 2014



Funded by:

Federal Ministry of Education and Research

Région de Bruxelles-Capitale

GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

This page is intentionally left blank

Work-Package 3: “Modeling”

OETCS/WP3/D3.5.1.1
September 2014

openETCS System Architecture and Design Specification

First Iteration: ETCS Kernel Functions

Document approbation

Lead author:	Technical assessor:	Quality assessor:	Project lead:
location / date	location / date	location / date	location / date
signature Bernd Hekele (DB-Netz)	signature [assessor name] ([affiliation])	signature Izaskun de la Torre (SQS)	signature Klaus-Rüdiger Hase (DB Netz)

Bernd Hekele, Peter Mahlmann, Peyman Farhangi

DB-Netz AG
Völckerstrasse 5
D-80959 München Freimann, Germany

Uwe Steinke

Siemens AG

Christian Stahl

TWT-GmbH

Architecture and Design Specification

Prepared for openETCS@ITEA2 Project

Abstract: This document gives an introduction to the architecture of the first openETCS iteration, the openETCS kernel functions. It has to be read as an add-on to the models in SysML, Scade and to additional reading referenced from the document.

Modification History

Version	Section	Modification / Description	Author
0.1	Document	Initial document providing the structure	Bernd Hekele
0.2	Sect. 3.4.3	initial contribution and some pretty printing	Christian Stahl
0.3	Document	collecting feedback and completion on initial sections	Bernd Hekele

Table of Contents

Figures and Tables

Figures

Tables

1 Introduction

1.1 Motivation

The openETCS work package WP3 aims to provide the architecture and the design of the openETCS OBU software as mainly specified in [1] UNISIG Subset_026 version_3.3.0.

The appropriate functionality has been divided into a list of functions of different complexity (see the WP3 function list [2]).

All these functions are object of the openETCS project and have to be analysed from their requirements and subsequently modelled and implemented. With limited manpower, a reasonable selection and order of these functions is required for the practical work that allows the distribution of the workload, more openETCS participants to join and leads to an executable—limited—kernel function as soon as possible.

While the first version of this document focuses on the first version of the limited kernel function, it is intended to grow in parallel to the growing openETCS software.

1.2 Objectives

The first objective of WP3 software shall be

- "Make the train run as soon as possible, with a very minimum functionality, and in the form of a rapid prototype."

This does not contradict the openETCS goal to conform to EN50128.

- After a phase of prototyping, the openETCS software shall be implemented in compliance to EN50128 for SIL4 systems.

Additional goals for this document are

- Identification of the functions required for a minimum OBU kernel
- Architecture overview regarding the minimum OBU kernel
- Technical approach: Description of the proceeding and methods to be used
- Road map of the minimum OBU kernel functions
- Road map thereafter

Note: This document will be extended according to the progress of WP3.

1.3 History

1.4 Goals of the openETCS Modelling Work

1.4.1 Functional Scope: The Minimum OBU Kernel Function

The objective “Make the train run with a very minimum functionality” shall be in terms of ETCS OBU translated into

- The Train moves on a track equipped with balises and determines its position.

That means, for this very first step, the train shall not supervise the maximum speed nor activate the brakes. The minimum function set shall be limited to:

- Receive, filter and manage balise information received from track (see `https://github.com/openETCS/SRS-Analysis/issues/12`)
- Calculate the actual train position based on balise and odometry information (see `https://github.com/openETCS/SRS-Analysis/issues/8`)
- Calculate the distances between the actual train position to track elements in its front

The activities of the first iteration are collected in the [3] openETCS WP3 backlog for the first iteration.

A more detailed architectural breakdown of these functions is available as a SysML model [4]. Diagrams used in this document describing the architecture are taken from this model.

The design is implemented in the [5] Scade Model. Design documents are taken from this model. Design diagrams used in this document are generated from the model. The design documents produced from the scade model are provided in the design location on Github [6].

In addition, the work on this minimum functionality requires to be supported by

- The availability of the ETCS language as specified in Subset UNISIG Subset_026, chapters 7 and 8
- The abiltiy to link intermediate and final results with the requirements of the ETCS specification (subset_026, . . .)

These supporting prerequisites are under construction and therefore not completely operable actually. How to deal with these restrictions, will be outlined in chapter ???

1.5 Glossary and Abbreviations

API Application Programming Interface
EVC European Vital Computer
BTM Balise Transmission Module
SRS System Requirements Specification

2 The openETCS Architecture of the initial kernel functions

2.1 The openETCS Tool-Chain and its impacts on the actual model

For understanding the modelling process and the modeling guidelines, we refer to [7].

To summarize the design process, the following rules are in use:

- Papyrus / SysML is used for modelling the architecture. Functions are visible on this SysML level.
- No behaviour model is allowed on SysML level.
- For referencing the requirements, links from the SysML model to the requirements document (in ProR) are being used.
- Details and especially behaviour is part of the Scade models.
- All interfaces (see also data-dictionary below) are available on bit-level.
- In the architecture model in SysML, all interfaces are available on a functional level for interfaces inside and outside the model and for interfaces between dedicated functions. Due to tool constrains the current model does not show all details for all interfaces (see dataDictionary).

The openETCS tool-chain for doing the modelling work consists of the following components:

Papyrus : for modelling the architecture (Kepler version).
In this phase only the Kepler version of the tool can be used due to incompatibilities of the Kepler and the Luna version on the SysML model. The SysML models are stored in the following location: `https://github.com/openETCS/modeling/tree/master/model/sysml`.

ProR : for keeping the requirements (REQIF).
The subset 26 is converted into a REQIF-format and also stored in the modeling repository on Github. The openETCS toolchain supports the linking of SysML model parts to SRS-Requirements. These results are also part of the architecture.

Scade : for designing and formalising the functions Scade version 15.2 is used.
The models are stored in this location: `https://github.com/openETCS/modeling/tree/master/model/Scade`. With the component Scade System Scade also has a component for designing the architecture.

In principle, the synchronisation mechanism of Scade was planned to be used for synchronising the SysML architecture and the Scade models. The idea is to automatically synchronise the SysML types and blocks with the Scade type definitions and the Scade Operators. Unfortunately, with the current set of tools this idea cannot be realised. We will investigate other tools and models to find a solution.

In addition, faults in the Kepler Papyrus version made it difficult for several members of the team to work on different submodels of the openETCS model. The issue will be solved when changing to the Luna version of Papyrus.

2.2 The openETCS Architecture

The following diagrams are taken from the SysML model [4].

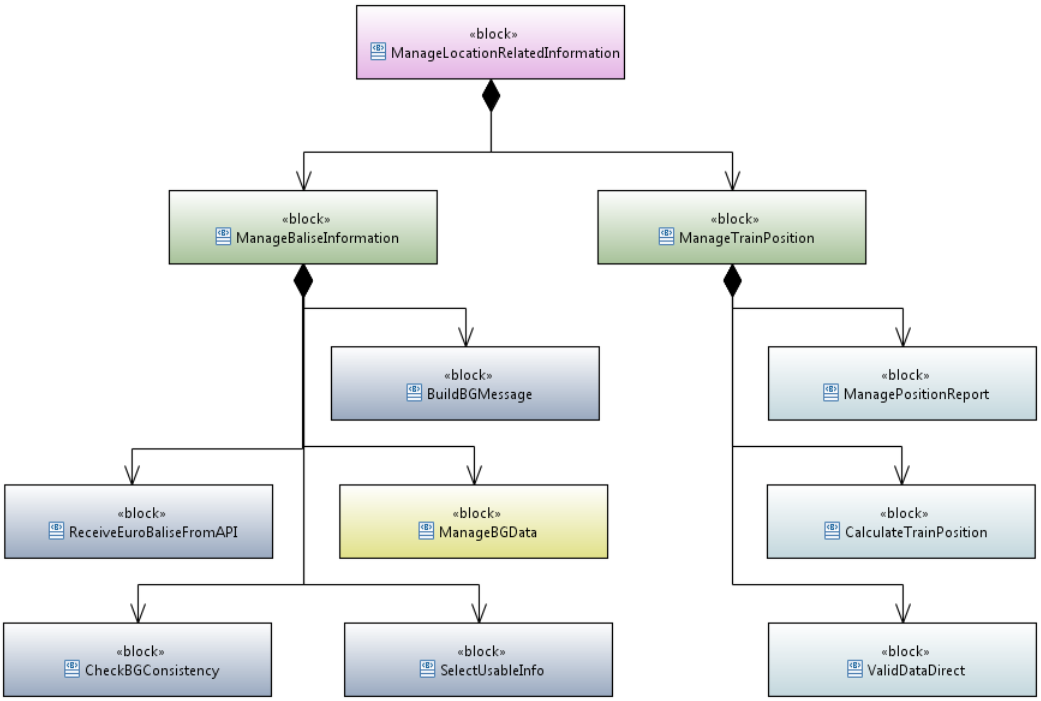


Figure 1. Block Definition Diagram of the First Iteration Architecture

The diagram shows the hierarchy of the EVC model. The boundaries of the model are given with the API (interfaces into and outside the EVC model), which actually is not part of the diagram. The runtime-system of the EVC is also seen as a part outside the model.

Green blocks in this diagram are seen as data collected by the "train" without making use of the function in focus.

Input to the model is via API. The border is inside the EVC Scade model.

3 openETCS Functions

3.1 openETCS Data Dictionary

The openETCS data Dictionary in the first iteration gives some basic constructs for the project, the definition of interfaces in a central place and the definition of the ETCS language [8].

3.1.1 ETCS Language

The typedefinitions of Subset 26 chapters 7 and 8 (called the ETCS language) are provided as SysML resp. Scade types to the openETCS model. For the SysML model the types have been generated based on tools and provided as <>package<> imported to the openETCS toolschain.

3.1.2 openETCS Interfaces

Interfaces used within the between submodels and interfaces from ouside and to outside the EVC kernel are defined as types in the dataDictionary.

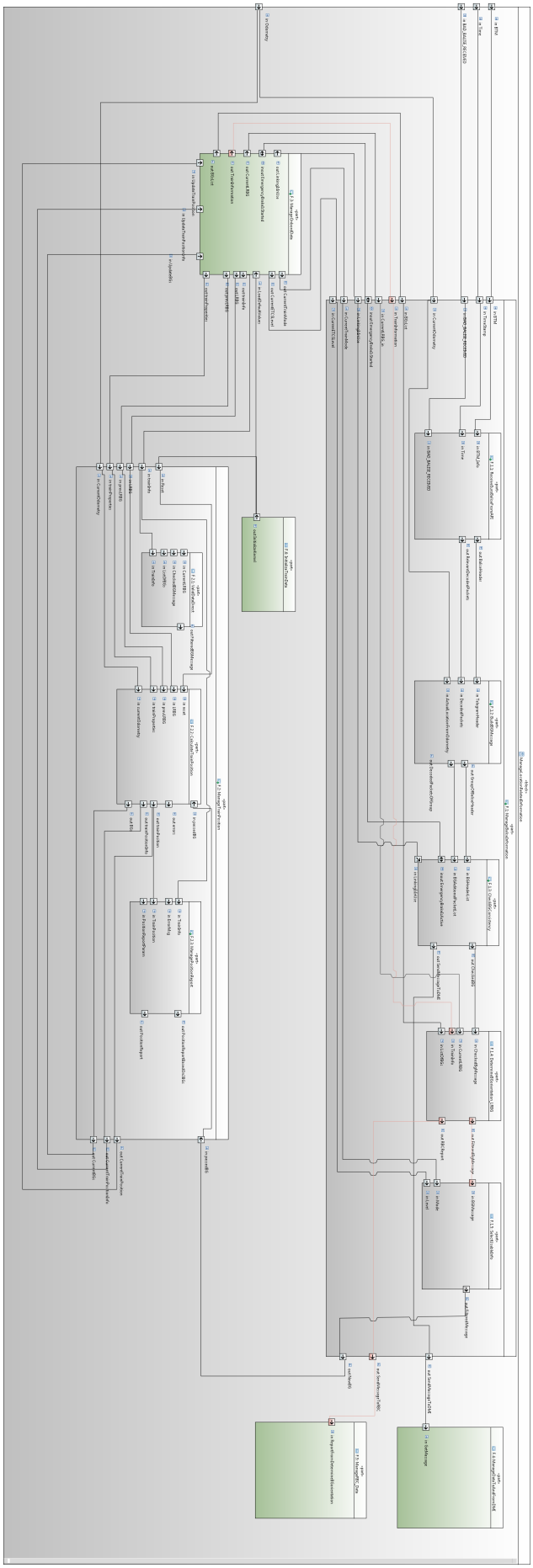


Figure 2. Internal Block Diagram of the First Iteration Architecture

In the Scade model the ETCS language is available in the oETCS projects S026-7 and S026-8.

3.1.3 dataDictionary Outlook

In the first iteration the use of the data dictionary concept is reduced to a minimum. The full openETCS process is tailored for a bigger team to cooperate and make use of tools to collect data and generate code.

In the scade model the types needed to build the ointerfaces between models are defined in the projects Obu_Basic_Types.etp, BG_Types.etp, and TrainPosition_Types.etp.

3.2 openETCS Generic API

3.2.1 Generic API

Because the API is currently not defined to that level, the following assumptions are implemented:

- **Eurobalise (BTM)**: One telegram per call ...

3.3 F.1 Manage Balise Information

3.3.1 F.1.1 Receive Eurobalise From API

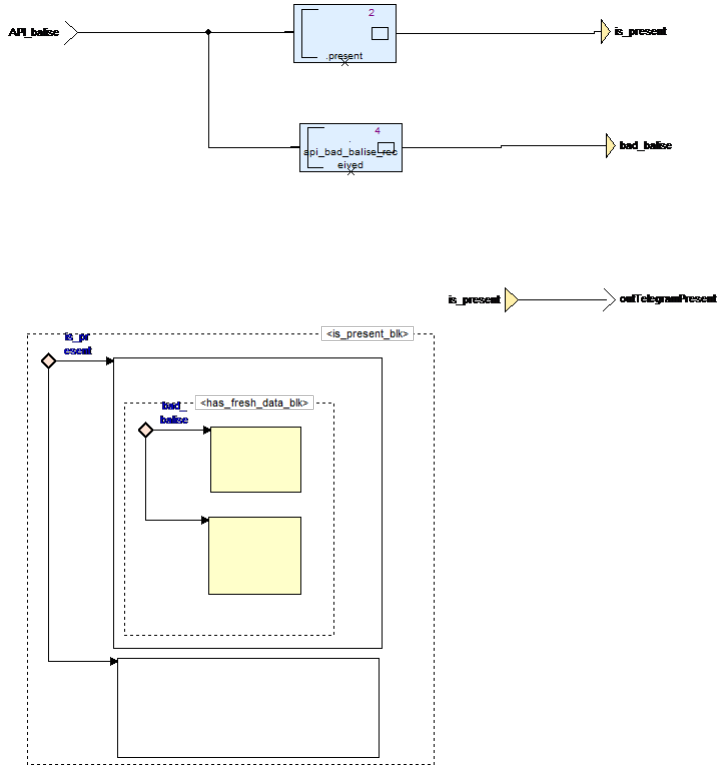


Figure 3. Structure of ReceiveEuroBaliseFromAPI

- **Short Description of Functionality**
  This function defines the interface of the OBU model to the openETCS generic API for Eurobalise Messages. On the interface, either a valid telegram is provided or a telegram is indicated which could not be received correct when passing the balise. The function passes

the telegram without major changes of the information to the next entity for collecting the balise group information.

- **Design Constrains and Choices**

  1. Decoding of balises is done at the API. Also, packets received via the interface are already transformed into a usable shape.
  2. Only packets used inside the current model are passed via the interface:
     Packet 5: Linking Information.
     Linking Information is filled into the linking array starting from index 0 without gaps. Used elements are marked as valid. Elements are sorted according to the order given by the telegram sequence.

3.3.2 F.1.2 Build BG Group Message

- **Short Description of Functionality**
  This entity collects telegrams received via the interface into Balise Group Information.
- **Reference to the SRS (or other requirements**
- **Design Constrains and Choices**

  1. Telegrams received as invalid are passed to the “Check-Function” in order to process errors in communication with the trackside according to the requirements and in a single place. Telegrams are filled into the telegram array starting from index 0 without gaps. Used elements are marked as valid. Elements are stored according to the order given by the telegram sequence.
  2. This function does not process information from the packets. The information is passed forward to the check without furhter processing of the values.

3.3.3 F.1.3 Check BG Consistency

- **Short Description of Functionality**

3.3.4 F.1.4 Determine BG- Orientation and LRBG

- **Short Description of Functionality**
- **Reference to the SRS (or other requirements**
- **Design Constrains and Choices**

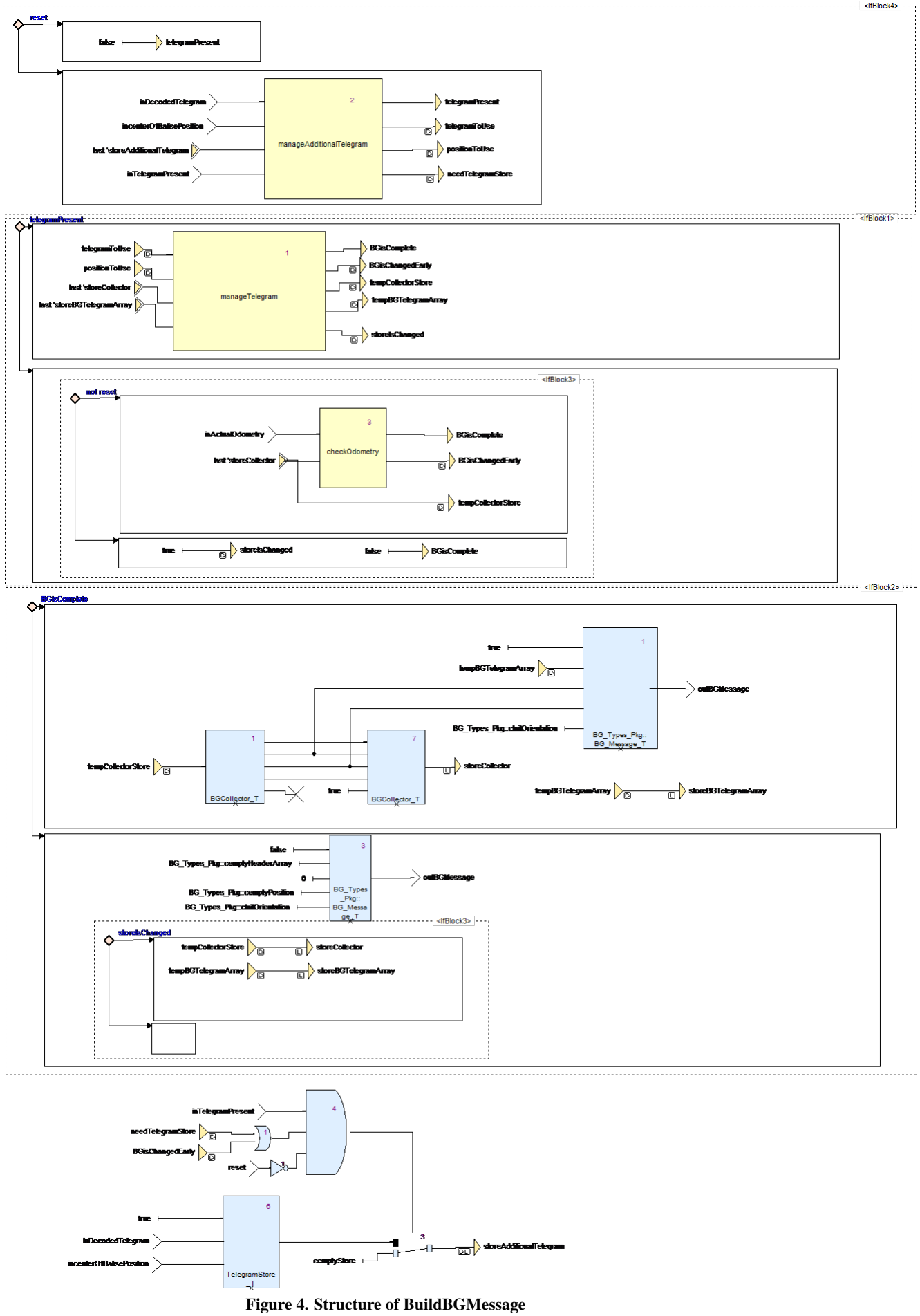


Figure 4. Structure of BuildBGMessage

3.3.5 F.1.5 Select Usable Info

3.4 F.2 Manage Train Position

3.4.1 F.2.1 Validate Data Direct

- **Short Description of Functionality**

- **Reference to the SRS (or other requirements**

- **Design Constrains and Choices**

3.4.2 F.2.2 Calculate Train Position

- **Short Description of Functionality**
  The main purpose of the function is to calculate the locations of linked and unlinked balise groups (BGs) and the current train position while the train is running along the track.

  #### Functional Structure in Stages

  The whole function calculateTrainPosition is subdivided into the following steps, which are performed sequentially:

  1. ***calculateBGLocations***: Calculate the balise group locations
     The first stage is triggered each time the train passes a balise group (input *passedBG*). It takes the balise group header with the BG identification, the linking information (Subset 26, packet 5) and the current odometry values as inputs and calculates the location of the the passed balise group. If the passed BG has been announced via linking information previously, it takes into account the linking as well as the odometry information. If the passed BG does not meet the tolerance window announced by linking, an error flag is set. If the passed BG is an unlinked BG, its location is determined by odometry only, but related to the next previously passed linked BG, if there is one.
     Then, if the passed BG is a linked BG comprising linking information for BGs ahead, the linking information is evaluated by creating the announced BGs and computing their locations from the linking distances.
     The passed and the announced BGs are stored in a list *BGs*, ordered by their nominal location on the track.
     Afterwards the locations of all BGs are further improved by re-adjusting their locations with reference to the just passed BG. This optimizes the BG location inaccuries around the current train position (= location of the passed BG).
  2. ***delDispensableBGs***: Delete dispensable balise groups
     The second stage removes balise groups supposed not to be needed any longer from the list of *BGs*.
     If the number of stored passed linked BGs exceeds the maximum number of eight as specified in subset-26-3.6.2.2.2 c), all BGs astern are deleted. If only (passed) unlinked BGs are in the list and exceed the number of *cNoOfAtLeast_x_unlinkedBGs*, all passed BGs astern to those are removed from the list.

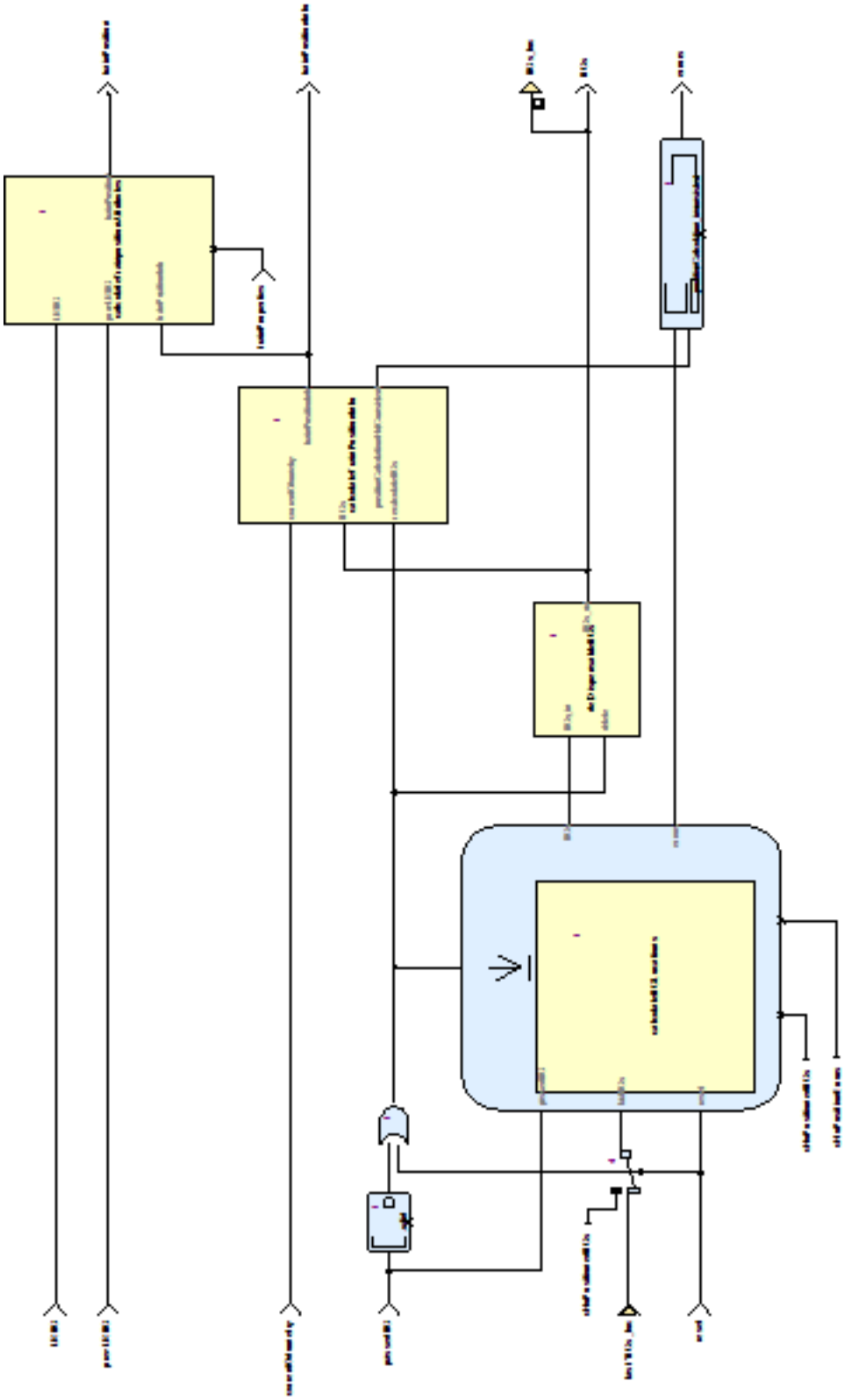


Figure 5. Structure of calculateTrainPosition

3. ***calculateTrainPositionInfo***: Calculate train position information.
   This stage take the list of stored BGs and the current odometry values as inputs and steadily provides the current train position.
4. ***calculateTrainpositionAttributes***: Calculate train position attribute information.
   This stage provides several additional position related attributes that might conveniently be used by subsequent consumers in the architecture. It requires the actual LRBG and the previous LRBG to be assigned external from the list *BGs*.

- **Reference to the SRS (or other requirements)**

  The component calculateTrainPosition determines the location of linked and unlinked balise groups and the current train position during the train trip as specified mainly in subset-026-3.6

- **Design Constraints and Choices**

  The following constraints and prerequisites apply:

  1. The input data received from the balises groups must have been checked and filtered for validity, consistency and the appropriate train orientation before delivering them to calculateTrainPosition.
  2. The storage capacity for balise groups is finite. calculateTrainPosition will raise an error flag when a balise group cannot be stored due to capacity limitations.
  3. calculateTrainPosition will raise an error flag if a just passed balise group is not found where announced by linking information. It will not (yet) detect when an announced balise group is missing.
  4. calculateTrainPosition is not yet prepared for train movement direction changes.
  5. calculateTrainPosition does not yet consider repositioning information.

3.4.3 Provide Position Report

- **Short Description of Functionality**
  This function takes the current train position and generates a position report which is sent to the RBC. The point in time when such a report is sent is determined from event, on the one hand, and position report parameters—which are basically triggers—provided by the RBC or a balise group passed, on the other hand. The functionality is modeled using three operations, as shown in Fig. 6, which are explained below.

  CalculateSafeTrainLength Calculates the the safeTrainLength according to Chapt. 3.6.5.2.4/5. `safeTrainLength = absolute(EstimatedFrontEndPosition - MinSafeRearEnd)`, where `MinSafeRearEnd = minSafeFrontEndPosition - L_TRAIN`

  EvaluateTriggerAndEvents Returns a Boolean modeling whether the sending of the next position report is triggered or not. It is the conjunction of the evaluation of all triggers (PositionReportParameters, i.e., Packet 58) and events (see Chapt. 3.6.5.1.4).

  CollectData In this operation, data of Packet0, . . . , Packet5 and the header is aggregated to a position report.

- **Reference to the SRS (or other requirements**
  Most of the functionality is described in subset 26, chapter 3.6.5.

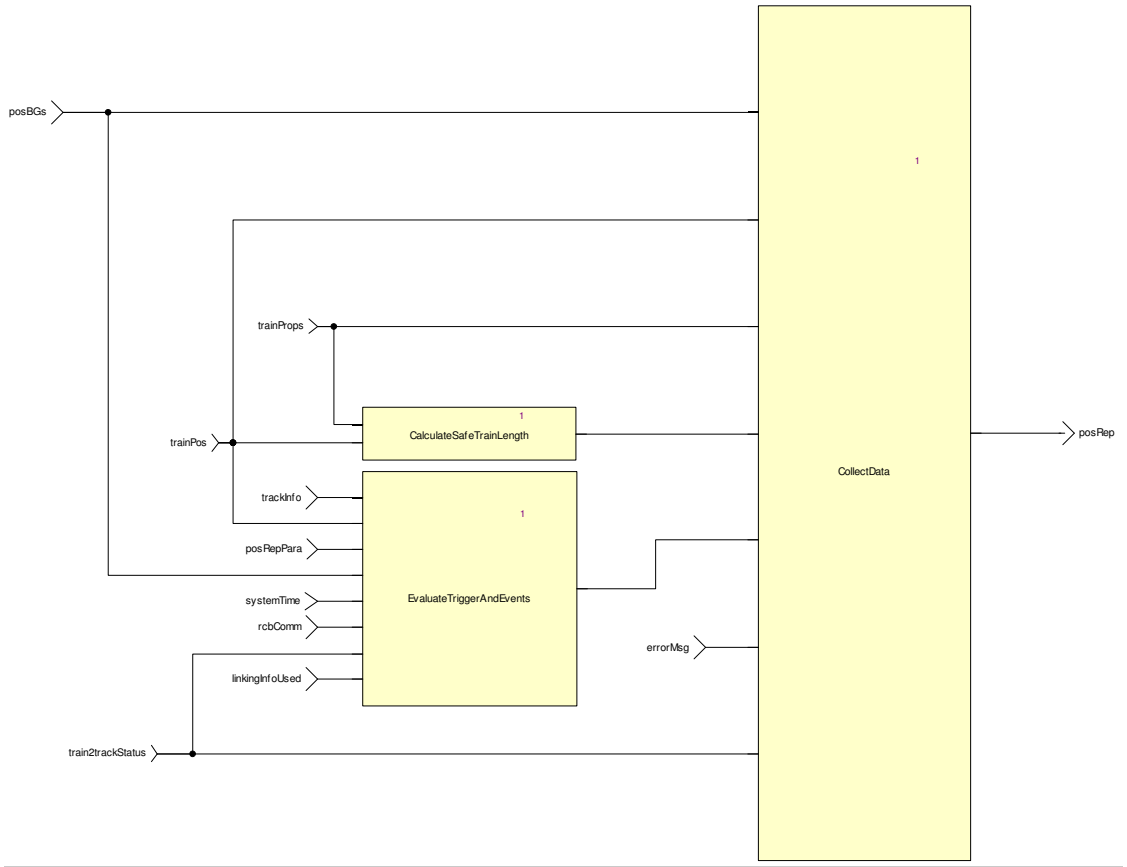


Figure 6. Structure of component ProvidePositionReport

- **Design Constrains and Choices**

  1. The message length (i.e., attribute `L_MESSAGE`) is by default set to 0; the actual value will be set by the Bitwalker/API.
  2. The attribute `Q_SCALE` is assumed to be constant; that is, all operations using this attribute do not convert between different values of that attribute.
  3. *PositionReportHeader*: The time stamp (i.e., attribute `T_TRAIN`) is not set; this should be done once the message is being sent by the API
  4. *Packet4*: When aggregating the data for this packet, an error message might be overwritten by a succeeding error message. Because the specification only allows to sent one error in one position report, errors are not being stored in a queue, for instance.
  5. *Packet44*: This packet is currently not contained in a position report as it is not part of the kernel functions.
  6. The usage of attributes `D_CYCLOC` and `T_CYCLOC` as part of the triggers specified by the position report parameters (i.e., Packet 58 sent by the RBC) may lead to unexpected results if a big clock cycle together with small values for the attributes is used. The cause is that the current model increments at every clock cycle the reference value for the distance and time by at most `D_CYCLOC` and `T_CYCLOC`, respectively and not a factor of it.

- **Open Issues**

  1. Operation *EvaluateTriggerAndEvents* currently ignores parameters `N_ITER`, `D_LOC` and `D_LGTLOC` which allow to specify up to 32 position at which a report has to be sent.

The positions are relative to the location of a reference balise group. If the RBC sends packet 58, then it also provides a reference balise group; otherwise, if packet 58 is sent by a balise group, then this balise group serves a the reference balise group. Possible realisation in the model: Extend in the interface posRepPara (i.e., Packet 58) by a `NID_BG` referring to the reference balise group. Am assumption would be that this BG can be found in the list of passed balise group provided by *CalculateTrainPosition* in Sect. 3.4.2.

2. The specification requires to store the last eight balise groups for which a position report has been sent (see 3.6.2.2.2.c).
3. For all reports that contain Packet 1 (i.e., report based on two balise groups), the RBC sends a coordinate system. It is unclear where this has to be stored (i.e., somehow the balise groups have to be stored in a database which has then to be updated), see 3.4.2.3.3.6. Moreover, such a coordination system can be invalid and then has to be rejected (see 3.4.2.3.3.7-8). On a more abstract level, we need to think about the interface between the RBC and the OBU or a proper abstraction thereof.
4. The decision whether a the report consists of packet 0 or packet 1, which is provided in 3.4.2.3.3, is currently not completely modeled. So far, 3.4.2.3.3.1 has only been modeled, thereby assuming "the last balise group detected" is the last balise group and not the LRBG. 3.4.2.3.3.2 is unclear. To model 3.4.2.3.3.4 I need information about the last two valid balise groups and the train running direction. This information can be obtained by adding a memory or this information will be provided by *CalculateTrainPosition* in Sect. 3.4.2. Likewise, also 3.4.2.3.3.5 requires knowledge about the last two valid balise groups.

References

[1] ERA. *System Requirements Specification, SUBSET-026*, v3.3.0 edition, March 2012.

[2] openETCS WP3-team. *openETCS Working Result: List of ETCS Functions*, November 2013. `https://github.com/openETCS/SRS-Analysis/blob/master/System%20Analysis/List_Functions.xlsx`.

[3] openETCS Product Owner. *WP3 Product Backlog, Selection of First Iteration*. `https://github.com/openETCS/SRS-Analysis/labels/backlog%20item`.

[4] openETCS. *openETCS SysML model*, 2014. `https://github.com/openETCS/modeling/tree/master/model/sysml/WP3-Initial-Architecture`.

[5] openETCS. *openETCS SCADE model*, 2014. `https://github.com/openETCS/modeling/tree/master/model/Scade/System`.

[6] openETCS WP3. *WP3 Design Documents: First Iteration*. `https://github.com/openETCS/modeling/blob/master/openETCSDesignDocuments/firstIteration/WP3_InitialArchitecture_DesignDescription.pdf`.

[7] Uwe Steinke Marielle Petit-Doche and Bernd Hekele. *WP3 Description of Work*, January 2014. `https://github.com/openETCS/modeling/blob/master/DescriptionOfWork/NewModelingDescriptionOfWork.pdf`.

[8] openETCS. *openETCS Model for the dataDictionary*, 2014. `https://github.com/openETCS/modeling/tree/master/model/sysml/dataDictionary`.