OETCS/WP3/D3.5.4-API

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

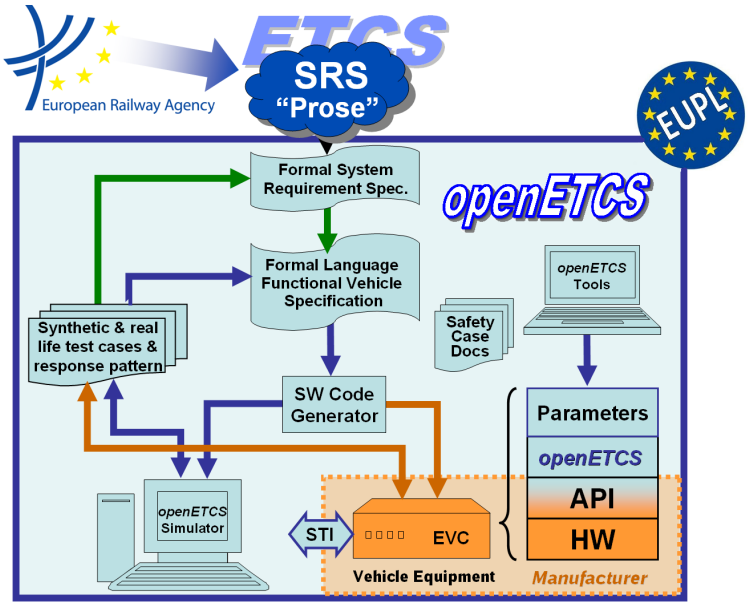
Work-Package 3: “Modeling”

openETCS API

Extension of the openETCS Architecture and Design Document

Bernd Hekele and David Mentré

November 2015



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/D3.5.4-API
November 2015

openETCS API

Extension of the openETCS Architecture and Design Document

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

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

David Mentré

Mitsubishi Electric R&D Centre Europe

Architecture Document

Prepared for openETCS@ITEA2 Project

Abstract: This document gives an introduction to the openETCS API.

Modification History

Version	Section	Modification / Description	Author
0.1	Document	Basic to this work is the documentation provided by David Mentré as a result of the openETCS API Team	David Mentré
0.2	Document	Update based on integration results	Bernd Hekele

Table of Contents

Figures and Tables

Figures

Tables

1 Introduction

The openETCS API is a major output of the openETCS project. It defines the interface to the openETCS EVC as an open document. It is based on one side on the industry based Alstom API definition documented in openETCS Requirements section D2.7.

On the other side it gives an introduction on how to interface to the EVC Scade model as the modelling result of openETCS project.

2 Input Documents

The implementation is based on subset-26 version 3.3.0 [1]. This version is part of the TSI [2].

https://github.com/openETCS/modeling/wiki/Input-Documents-Repository

The openETCS Alstom API is documented in the openETCS Requirements repository. It consists of the following parts: [3] API Requirements [4] API DataDictionary [5] API Application Layer

This document is basede on the openETCS API team work results. The complete set of work results is available in this repository: `https://github.com/openETCS/modeling/tree/master/API`

3 openETCS Architecture with respect of the API

3.1 Abstract Hardware Architecture

For proper understanding of openETCSAPI and of constraints imposed on both sides of the API, we need to define a *reference abstract hardware architecture*. This hardware architecture is "abstract" is the sense that the actual vendor specific hardware architecture might be totally different of the abstract architecture described in this chapter. For example, several units might be grouped together on the same processor.

However the actual vendor specific architecture shall fulfil all the requirements and constraints of this reference abstract hardware architecture and shall not request additional constraints.

3.2 Definition of the reference abstract hardware architecture

The reference abstract hardware architecture is shown in figure 1.

The reference abstract hardware architecture is made of a bus on which are connected *units* defining the OBU:

EVC : European Vital Computer, the Unit where ETCS train control is running.

TIU : Train Interfacing Unit

ODO : The odometery system of the train.

DMI : The driver machine interface.

STM : The Specific Transmission Module. This interface is not in the focus of the openETCS project.

BTM : The Balise Transmission Module, a receiver hosted on the train for receiving telegrams sent from track to the train.

LTM : The Loop Transmission Module. Not part of this openETCS implementation;

EURORADIO : The interface hosted in the train for sending and receiving messages between Train and RBC in both directions. The module receiving and sending messages is called

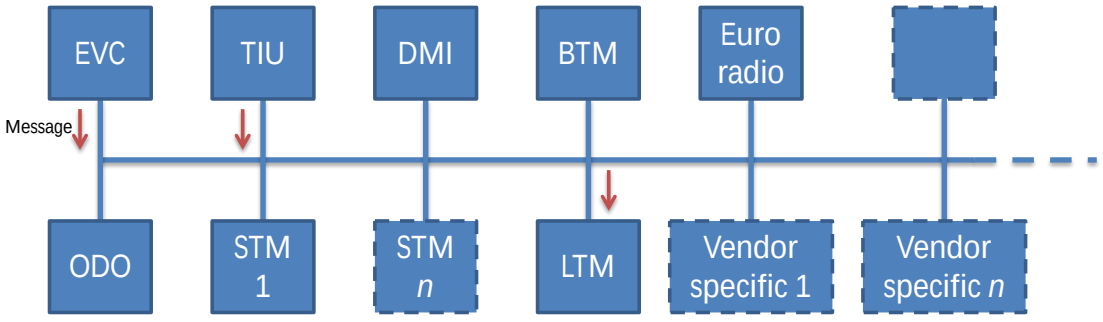


Figure 1. Reference abstract hardware architecture

radio transmission module [RTM]. The train may be equipped with up to 2 radio connections for machine to machine communication in an ETCS train.

JRU : Not part of this openETCS implementation;

Elements not being part of this implementation are marked.

Those units shall be working concurrently. They shall exchange information with other units through asynchronous message passing.

3.3 Reference abstract software architecture

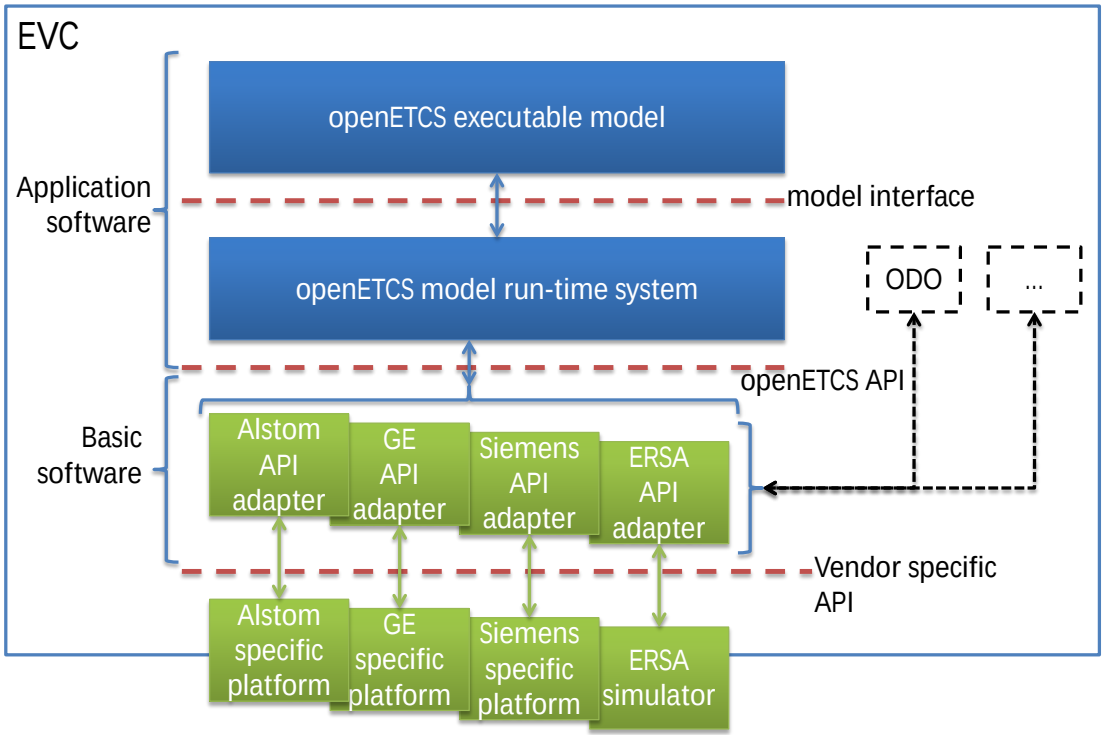


Figure 2. Reference abstract software architecture

The *reference abstract software architecture* is shown in figure 2. This architecture is made of following elements:

- *openETCS executable model* produced by the [6] Scade Model. It shall contain the program implementing core ETCS functions;
- *openETCS model run-time system* shall help the execution of the openETCS executable model by providing additional functions like encode/decode messages, proper execution of the model through appropriate scheduling, re-order or prioritize messages, etc.
- *Vendor specificAPI adapter* shall make the link between the Vendor specific platform and the openETCS model run-time system. It can buffer message parts, encode/decode messages, route messages to other EVC components, etc.
- All above three elements shall be included in the EVC;
- *Vendor specific platform* shall be all other elements of the system, bus and other units, as shown in figure 1.

We have thus three interfaces:

- *model interface* is the interface between openETCS executable model and openETCS model run-time system.
- *openETCSAPI* is the interface between openETCS model run-time system and Vendor specificAPI adapter.
- *Vendor specificAPI* is the interface between Vendor specificAPI adapter and Vendor specific platform. This interface is not publicly described for all vendors. You can find the Alstom imnplementation as an example.

The two blocks openETCS executable model and openETCS model run-time system are making the *Application software* part. This Application software might be either openETCS reference software or vendor specific software.

The Vendor specificAPI adapter is making the *Basic software* part.

3.4 Principles for Interfaces (openETCS_API)

Information is exchanged as *messages* in an asynchronous way. A message is a set of information corresponding to an event of a particular unit, e.g. a balise received from the BTM.

The information is passed to the executable model as parameters to the synchronous call of a procedure (Interface to the executable model). Since the availability of input messages to the application is not guaranteed, the parts of the interfaces are defined with a "present" flag. In addition, fields of input arrays are quite often of variable size. Implementation in the concrete interface in this use-case is the use of a "size" parameter and a "valid"-flag.

3.5 openETCS Model Runtime System

The openETCS model runtime system also provides:

- Input Functions From other Units
  In this entity messages from other connected units are received.
- Output Functions to other Units
  The entity writes messages to other connected units.
- Conversation Functions for Messages (Bitwalker)
  The conversion function are triggered by Input and Ouput Functions. The main task is to convert input messages from an bit-packed format into logical ETCS messages (the ETCS language) and Output messages from Logical into a bit-packed format. The logical format of the messages is defined for all used types in the openETCS data dictonary.
  Variable size elements in the Messages are converted to fixed length arrays with an used elements indicator.
  Optional elements are indicated with an valid flag. The conversion routines are responsible for checking the data received is valid. If faults are detected the information is passed to the openETCS executable model for further reaction.
- Model Cycle

The version management function is part of the message handling. This implies that conversions from other physical or logical layouts of messages are mapped onto a generic format used in the EVC. Information about the origin version of the messages is part of the messages.

The executable model is called in cycles. Cycle time is set to 10 msec. In the cycle:

- First the received input messages are decoded;
- The input data is passed to the executable model in a predefined order. The order is defined by the generated Scade model.
- Output of the executable model is encoded according to the SRS and passed to the buffers of the units.

4 Definition of the openETCS API

This section gives a short introduction of the openETCS EVC model interfaces. In addition, it gives an link to the ports of the EVC Scade model and the defining interface types. All descriptions refer to the EVC - Scade model which is located in this part of the repository:

`https://github.com/openETCS/modeling/blob/master/model/Scade/System/OBU_PreIntegrations/openETCS_EVC/openETCS_EVC.etp`

4.1 Interfaces to and from Track [BTM, RTM <-> EVC]

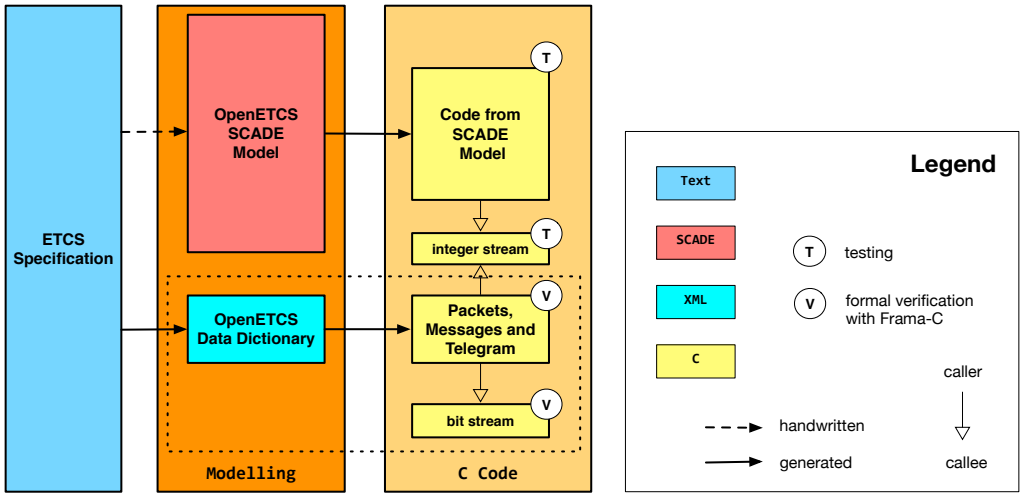


Figure 3. openETCS API Highlevel View

This part describes the interfaces resulting from BTM and RTM modules. In the Basic Software described above the following layers are defined for preparing received messages and telegrams for processing in the EVC resp. for preparing messages to the RBC for sending.

4.1.1 Physical Layer: Bitwalker

According to the SRS, messages and telegrams of the protocol between track (Balises and RBC) and train are defined on a densely packed and for transport purposes optimised structure. Information is passed in streams of bytes hiding the details.

This densely packed stream of data is first transformed to a stream of integers with an equivalent definition of the structure of the interfaces.

A major task of this layer is to guarantee errors induced while transporting the data is detected, e.g., by the BTM or RTM modules.

The implementation tailored for this task resides in the openETCS dataDictionary `https://github.com/openETCS/dataDictionary` project. This implementation represents the presentation of subset 26 sections 7 and 8 of the SRS in a physical definition.

4.1.2 Logical Layer: Bytewalker

In a second step the information provided by the Bitwalker is prepared for the input to the EVC. This function is implemented in the EVC Scade model by means of the track-messages function:

`https://github.com/openETCS/modeling/blob/master/model/Scade/System/ObuFunctions/ETCS_Messaging/TrackMessages/TrackMessages.etp`

This function is responsible for performing a first set of simple logical checks on the transformed information.

On the output from scade model to the RTM information is managed with the openETCS output bus, also being implemented in the track messages function.

4.1.3 Interfaces to the Scade Model

Having passed Bitwalker and Bytewalker the information coming from RTM and BTM is combined to a single interface to the EVC model.

The function is based on an input- buffer for track- messages. In this buffer, the messages of balises (from BTM) and of radio communication (from RTM) are sequenced in a single buffer. This buffer is interfacing to this evc.

Similarly, on the output side, a buffer is used for managing with multiple output messages to the radio interface in each cycle.

Input: API_fromTrack
Type: API_Msg_Pkg::API_TrackSideInput_T

Another interface is defined to give the status of the radio mobiles known to the EVC:
Input: API_mobileHWStatus
Type: MoRC_Pck::mobileHWStatus_Type_T

In the output direction information is passed from the EVC model to the RTM.

Output: API_toRBC
Type: TM_radio_messages::M_TrainTrack_Message_T

For controlling the radio modules a second interface is defined:
Output: API_RTM_management
Type: API_RadioCommunication_Pkg::RadioManagement_T

4.2 Interfaces to the DMI

The interfaces and protocols in the interface to the DMI is based on the protocol defined by ERSA in its DMI implementation. Details are not opensource.

Input: API_fromDMI
Type: API_DMI_Pkg::DMI_to_EVC_Message_int_T

Output: API_toDMI
Type: API_DMI_Pkg::EVC_to_DM_Message_int_T

Model: `https://github.com/openETCS/modeling/blob/master/model/Scade/System/APITypes/APITypes.etp`

4.3 Interfaces to the Time System

The interface types are defined in the OBU_Basic_Types_Pkg Package. The system time is defined in the basic software.

The system TIME is provided to the executable model at the begin of the cycle. It is not refreshed during the cycle. The time provided to the application is equal to 0 at power-up of the EVC (it is not a "UTC time" nor a "Local Time"), then must increase at each cycle (unit = 1 msec), until it reaches its maximum value (i.e current EVC limitation = 24 hours)

Input: API_SystemTime
Type: Obu_BasicTypes_Pkg::T_internal_Type

- TIME (T_internal_Type, 32-bit INT)
  Standardized system time type used for all internal time calculations: in ms. The time is defined as a cyclic counter: When the maximum is exceeded the time starts from 0 again.

Model: `https://github.com/openETCS/modeling/blob/master/model/Scade/System/ObuFunctions/Obu_BasicTypes/Obu_BasicTypes.etp`

4.4 Interfaces to the Odometry System

The interface types are defined in the OBU_Basic_Types_Pkg Package. The odometer gives the current information of the positing system of the train. In this section the structure of the interfaces are only highlighted. Details, including the internal definitions for distances, locations speed and time are implemented in the package.

Input: API_Odometry
Type: Obu_BasicTypes_Pkg::odometry_T

- Odometer (odometry_T)
  - valid (bool)
    valid flag, i.e., the information is provided by the ODO system and can be used.
  - timestamp (T_internal_Type)
    of the system when the odometer information was collected. Please, see also general remarks on the time system.
  - Coordinate (odometryLocation_T)
    - nominal (L_internal_Type) [cm]
    - min (L_internal_Type) [cm]
    - max (L_internal_Type) [cm]

The type used for length values is a 32 bit integer. Min and max value give the interval where the train is to be expected. The boundaries are determined by the inaccuracy of the positioning system. All values are set to 0 when the train starts.

- speed (V_internal_Type) [cm/sec] General Speed of the train
- acceleration (A_internal_Type)[0.01 m/s2],
  Standardized acceleration type for all internal calculations
- motionState (Enumeration)
  indicates whether the train is in motion or in no motion
- motionDirection (Enumeration)
  indicates the direction of the train, i.e., CAB-A first, CAB-B first or unknown.

Model: `https://github.com/openETCS/modeling/blob/master/model/Scade/System/ObuFunctions/Obu_BasicTypes/Obu_BasicTypes.etp`

4.5 Interfaces to the Train Interfaces (TIU)

The following information is based on the implementation of the Alstom API. The interface is organised in packets. Details on the interfaces can be seen from the Scade models. The interface is used on the input side to docuemnt the status of the train's units. On the output side, commands to the train units, especially the brake system, are available.

Input: API_fromTIU
Type: API_TIU_Pkg::TIU_Input_msg

Output: API_toTIU
Type: API_TIU_Pkg::TIU_Output_msg

Model: `https://github.com/openETCS/modeling/blob/master/model/Scade/System/APITypes/APITypes.etp`

References

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

[2] ERA. *New Annex A for ETCS Baseline 3 and GSM-R Baseline 0*, recommendation basleine 3 edition, April 17th 2012. http://www.era.europa.eu/Document-Register/Pages/New-Annex-A-for-ETCS-Baseline-3-and-GSM-R-Baseline-0.aspx.

[3] Nicolas Boverie. *API Requirements for OpenETCS*. Alstom Transport, v1.4 edition, September 2014. https://github.com/openETCS/requirements/blob/master/D2.7-Technical_Appendix/OETCS_API%20Requirements_v1.4.pdf.

[4] Alstom Transport. *Appendix Functional Data Dictionary*, v1.1 edition, 2014. https://github.com/openETCS/requirements/blob/master/D2.7-Technical_Appendix/OETCS_API%20Requirements_appendix_functional_data_dictionary_v1.1.pdf.

[5] Alstom Transport. *Appendix application layer*, v1.2 edition, 2014. https://github.com/openETCS/requirements/blob/master/D2.7-Technical_Appendix/OETCS_API%20Requirements_appendix_application_layer_v1.2.pdf.

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