OETCS/WP3/D3.5.4-API

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

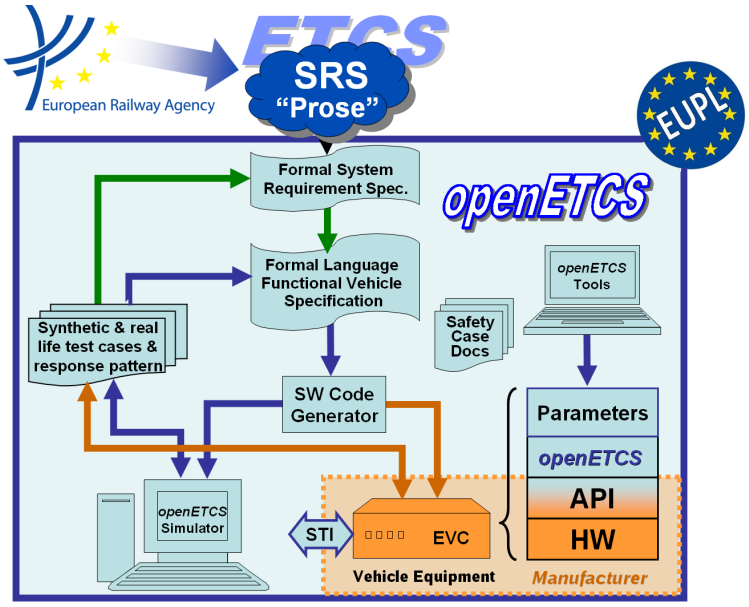
Work-Package 3: “Modeling”

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 and Functional Specification

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 Mentre as a result of the openETCS API Team	David Mentre
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 Introduction to the Architecture

3.1 Abstract Hardware Architecture

For proper understanding of openETCS API 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 on-board unit (OBU):

- European Vital Computer (EVC);
- Train Interface Unit (TIU);
- odometry (ODO);
- Driver Machine Interface (DMI);
- Specific Transmission Module (STM);
- Balise Transmission Module (BTM);
- Loop Transmission Module (LTM): Not part of this openETCS implementation;
- EURORADIO;
- Juridical Recording Unit (JRU): Not part of this openETCS implementation;

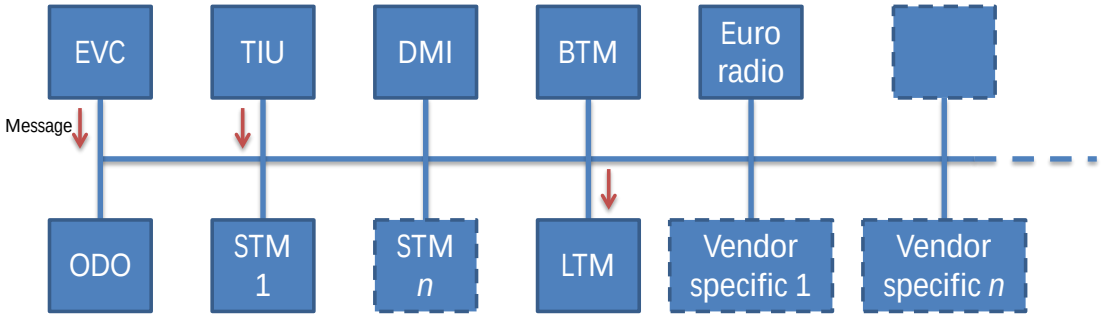


Figure 1. Reference abstract hardware architecture

Elements not being part of this implementation are marked.

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

3.3 Reference abstract software architecture

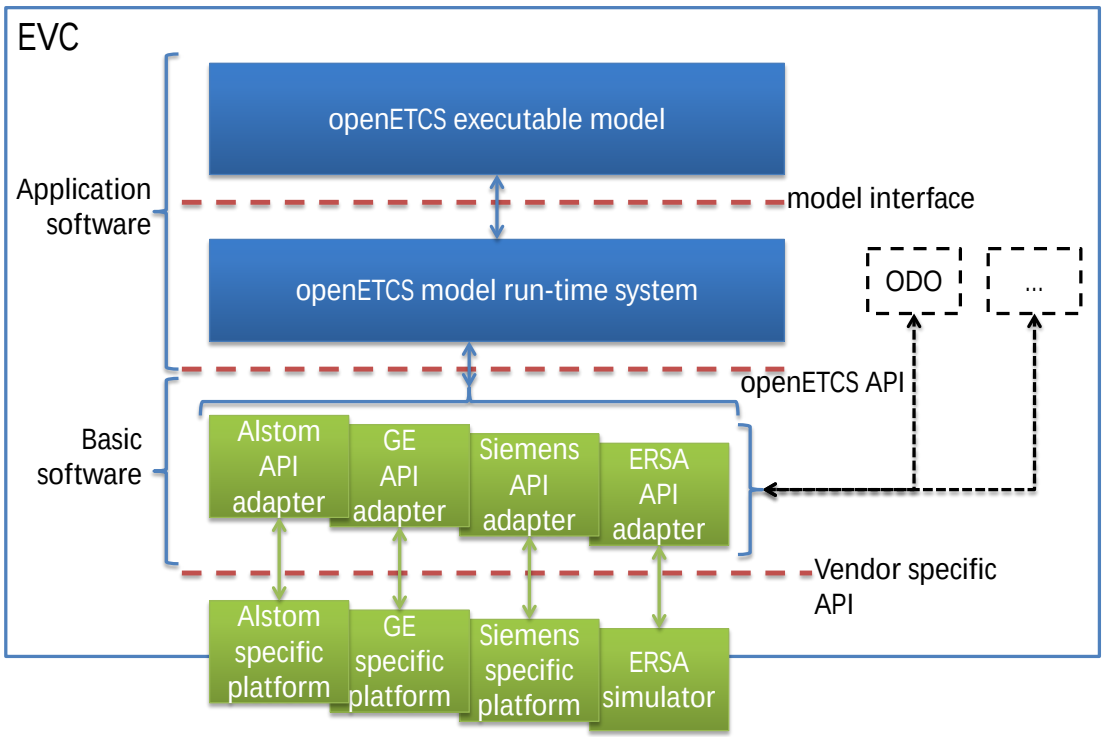


Figure 2. Reference abstract software architecture

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 specific API 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.

- *openETCS API* is the interface between openETCS model run-time system and Vendor specific API adapter.
- *Vendor specific API* is the interface between Vendor specific API 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 specific API adapter is making the *Basic software* part.

3.4 Interfaces to and from the Track

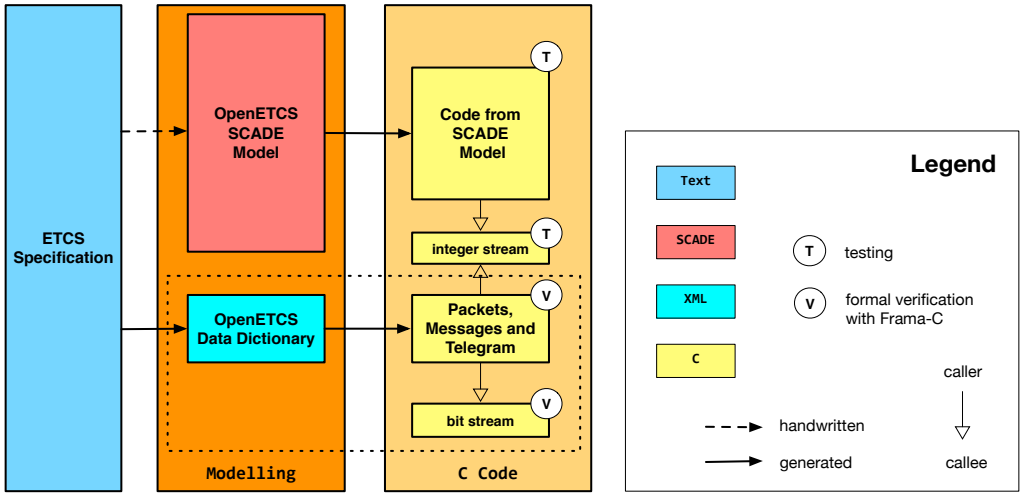


Figure 3. openETCS API Highlevel View

3.4.1 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 structue. 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.

3.4.2 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.

This function is responsible for logical checks on the transferred information.

3.5 openETCS API Runtime System and Input to the EVC)

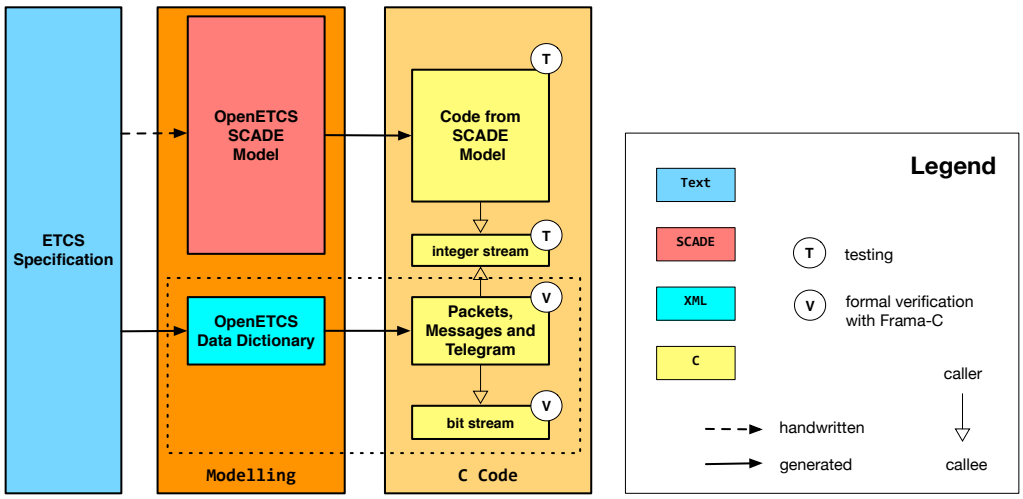


Figure 4. openETCS API Highlevel View

Figure 4 shows the structure of API with respect of the software architecture. Input boxes and output boxes not implemented in this stage are marked as red, other interfaces are marked as green. The System covers functions for processing Inputs from other Units, functions for processing Outputs to other functions and a basic runtime system. Inputs are used to feed the input to the executable model before calling it, outputs are used for collecting information provided by the executable model to be passed to the relevant interfaces after the execution cycle has finished.

3.6 Principles for Interfaces (openETCS API)

Information is exchanged *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 possible kind of messages are described in chapter **??**.

The information is passed to the executable model as parmeters to the snychronous 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 arraysquite often is of variable size. Implementation in the concrete interface in this use-case is the use of a "size" parameter and a "valid"-flag.

3.6.1 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, 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 message is part of the messages.

  The executable model is called in cycles. In the cycle

  – First the received input messages are decoded
  – The input data is passed to the executable model in a predefined order. **(Details for the interface to be defined)**.
  – Output is encoded according to the system requirement specification (SRS) and passed to the buffers to the units.

3.6.2 Input Interfaces of the openETCS API to other Units of the OBU

Interfaces are defined in the Scade project APITypes (package API_Msg_Pkg.xscade).

In the next table we can see the interfaces being used in the openETCS system. Details on the interfaces are defined further down.

Unit	Name	Processing Function	Description
BTM	Balise Telegram	Receive Messages	
DMI			
EURORADIO	Communication Management	Communication Management	
EURORADIO	Radio Messages	Receive Messages	
ODO	Odometer	All Parts	
TIME	Time system of the OBU	All Parts	
Startup			
TIU	Train Data	All Parts	

3.6.3 Output Interfaces of the openETCS API TO other Units of the OBU

From Function	Name	To Unit	Description
	Radio Output Message	EURORADIO	
	Communication Management	EURORADIO	
	Driver Information	DMI	
	Train Data	TIU	

Information in the following sections gives an more detailed overview of the structure of the interfaces.

3.6.4 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.

3.6.5 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)

- 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.

3.6.6 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.

- 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 bounderies are determined by the inaccuracy of the positioning system. All values are set to 0 when the train starts.
  - speed (V_internal_Type) [km/h] General Speed of the train
  - acceleration (A_internal_Type)[0.01 m/s2],
    Standardized acceleration type for all internal calculations : in

- 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.

3.6.7 Interfaces to the Train Interfaces (TIU)

The following infomration is based on the implementation of the Alstom API. The interface is organised in packets. The packets of the Alstom implementation are listed in the appendix to this document.

The description of interfaces needed for the current scope will be added according to the use.

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.