Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

From Commercial Documents to System Requirements: an Approach for the Engineering of Novel CBTC Solutions

Alessio Ferrari[1], **Giorgio O. Spagnolo**[1],
Giacomo Martelli[2], **Simone Menabeni**[2]

[1] ISTI-CNR, Via G. Moruzzi 1, Pisa, ITALY,
e-mail: {lastname}@isti.cnr.it,
WWW home page: http://www.isti.cnr.it/
[2] DSI, Università degli Studi di Firenze, Via di S.Marta 3, Firenze, ITALY,
e-mail: {lastname}@dsi.unifi.it,
WWW home page: http://www.dsi.unifi.it/

Received: date / Revised version: date

Abstract. Communications-based Train Control (CBTC) systems are the new frontier of automated train control and operation. Currently developed CBTC platforms are actually very complex systems including several functionalities, and every installed system, developed by a different company, varies in extent, scope, number, and even names of the implemented functionalities. International standards have emerged, but they remain at a quite abstract level, mostly setting terminology.

This paper presents the results of an experience in defining a global model of CBTC, by mixing *semi-formal modelling* and *product line engineering*. The effort has been based on an in-depth market analysis, not limiting to particular aspects but considering as far as possible the whole picture. The paper also describes a methodology to derive novel CBTC products from the global model, and to define system requirements for the individual CBTC components. To this end, the proposed methodology employs scenario-based requirements elicitation aided with rapid prototyping. To enhance the quality of the requirements, these are written in a constrained natural language (CNL), and evaluated with natural language processing (NLP) techniques. The final goal is to go toward a *formal* representation of the requirements for CBTC systems.

The overall approach is discussed, and the current experience with the implementation of the method is presented. In particular, we show how the presented methodology has been used in practice to derive a novel CBTC architecture.

Send offprint requests to:

Introduction

Communications-based Train Control (CBTC) is the most recent technological frontier for signalling and train control in the metro market [38,31]. CBTC systems offer flexible degrees of automation, from enforcing control over dangerous operations acted by the driver, to the complete replacement of the driver role with an automatic pilot and an automatic on-board monitoring system. Depending on the specific installation, different degrees of automation might be required. Furthermore, companies shall be able to provide complete CBTC systems, but also subsets of systems. The aim is to satisfy the needs of green-field installations, and address the concerns of the operators who wish to renew only a part of an already installed system. In this sense, the *product line engineering* technology provides a natural tool to address the need for modularity required by a market of this type [14,19].

Entering the CBTC market with a novel product requires such a product to be compliant with the existing *standards*. Two international standards provide general requirements for CBTC systems. The first is IEEE 1474.1-2004 [31], while the second is IEC 62290 [1,2]. The IEEE standard treats the CBTC system as a composition of sub-systems. Instead, the IEC standard look at the CBTC system as a whole, and considers the different Grades of Automation (GoA) that a CBTC system can achieve. In general, the standards differ in terminology and structure. Therefore, a product satisfying the former is not ensured to accomplish also the requirements of the latter.

Railway and metro system developed for Europe shall be also compliant with the CENELEC standards [11,10, 12]. This is a set of norms and methods to be used while implementing a product having a determined safety-critical nature. Besides *product-level* standard, a CBTC

product is therefore required to satisfy also *process-level* standards (i.e., the CENELEC norms).

The challenges related to the introduction of a novel CBTC system are not limited to the adherence to the standards. Indeed, also the *competitiveness* of the product plays a crucial role. To be competitive with the solutions of other vendors, a novel CBTC product shall take into account the existing similar products and installations. The CBTC market is currently governed by seven main vendors, namely Bombardier [46], Alstom [45], Thales [50], Invensys Rail Group [32], Ansaldo STS [3], Siemens [44], and GE Transportation [28]. Each vendor provides its own solution, and different technologies and architectures are employed.

In this paper an experience is presented, where domain analysis has been used to derive a global CBTC model, from which specific product requirements for novel CBTC systems can be derived. The global model is built upon the integration of the guidelines of the product-level standards, and is driven by the architectural choices of the different vendors. The model is represented in the form of a *feature diagram* [34,4,15], following the principles of the product-line engineering technology. From the global feature diagram, we derive the actual product requirements. To this end, we draw graphical formal models of the product architecture, together with scenario models in the form of simplified sequence diagrams. Architecture and scenario models are used to define and enrich the natural language requirements of the actual product.

After the definition of the product requirements, we define requirements for the individual systems that compose the CBTC product. To this end, we employ scenario-based requirements elicitation [47], aided with rapid prototyping [29]. A constrained natural language and natural language processing techniques [18] are used to evaluate and enhance the quality of the system requirements. The approach is oriented to satisfy the guidelines of the CENELEC standards for system requirements. A transition from the constrained natural language to a formal representation of the requirements is also foreseen.

Examples are presented throughout the paper to explain the approach, and to show the results of the current implementation of the proposed methodology.

The paper is structured as follows. In Sect. 1, the CBTC operational principles are presented. In Sect. 2, an overview of the approach is given. In Sect. 3, an analysis of the standards and of the architectures of the CBTC vendors is presented. In Sect. 4, the global CBTC model is described. In Sect. 5, the architecture and scenario models are derived, together with the requirements for the actual product. In Sect. 6, the approach for the definition of the requirements for the individual systems that compose the CBTC product is presented. Sect. 7 describes the current experience with the implementation of the method, and the lessons learnt. In Sect. 8, related works are discussed. Sect. 9 draws final conlusions and remarks.

1 Communications-based Train Control Systems

CBTC systems [38,31] are novel signalling and control platforms tailored for metro. These systems provide a continuous automatic train protection as well as improved performance, system availability and operational flexibility of the train.

The conventional metro signalling/control systems that do not use a CBTC approach are exclusively based on track circuits and on wayside signals. Track circuits are used to detect the presence of trains. Wayside signals are used to ensure safe routes and to provide information to the trains. Therefore, the position of the train is based on the accuracy of the track circuit, and the information provided to the train is limited to the one provided by the wayside signals. These systems are normally referred as *fixed block* systems, since the distance between trains is computed based on fixed-length sections (i.e., the length of a track circuit - see upper part of Figure 1).

CBTC overcomes these problems through a continuous wayside-to-train and train-to-wayside data communication. In this way, train position detection is provided by the onboard equipment with a high precision. Furthermore, much more control and status information can be provided to the train. Currently, most of CBTC systems implement this communication using radio transmission [35].

The fundamental characteristic of CBTC is to ensure a reduction of the distance between two trains running in the same direction (this distance is normally called *headway*). This is possible thanks to the *moving block* principle: the minimum distance between successive trains is no longer calculated based on fixed sections, as occurs in presence of track circuits, but according to the rear of the preceding train with the addition of a safety distance as a margin. This distance is the limit distance (MA, Movement Authority) that cannot be shortened by a running train (see lower part of Figure 1).

The control system is aware at any time about the exact train position and speed. This knowledge allows the onboard ATP (Automatic Train Protection) system to compute a dynamic braking curve to ensure safe separation of trains, which guarantees that the speed limit is not exceeded. The ATP system ensures that the MA is not shortened by the train, in addition to the continuous protection of the train in every aspect.

From the architectural point of view, CBTC systems are characterized by a division in two parts: onboard equipment and wayside equipment. The first is installed on the train and the latter is located at a station or along the line.

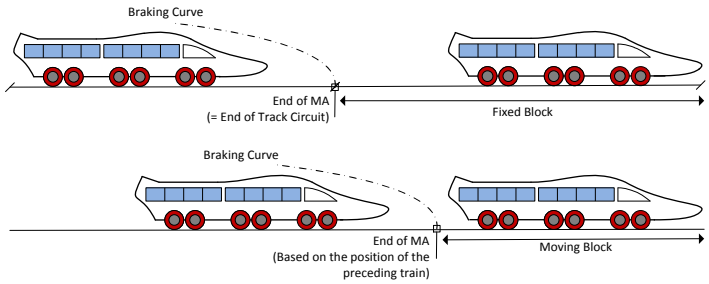


Fig. 1: Fixed block *vs* moving block

CBTC systems also allow automatic train control functions by implementing both the ATO (Automatic Train Operation) and the ATS (Automatic Train Supervision) functionalities. The ATO enables driverless operation, ensuring the fully automatic management of the train in combination with ATP. The ATS offers functions related to the supervision and management of the train traffic, such as adjustment of schedules, determination of speed restrictions within certain areas and train routing.

A CBTC system might include also one or more interlocking (noted in the following as IXL). The IXL monitors the status of the objects in the railway yard (e.g., switches, track circuits) and, when routing is required by the ATS, allows or denies the routing of trains in accordance to the railway safety and operational regulations.

2 Method Overview

In this work an approach has been defined to identify a global model of CBTC and derive the product requirements for a novel CBTC system. The method starts from the available international requirements standards – IEEE 1474.1-2004 [31] and IEC 62290 [1,2] – and from the public documents provided by the current CBTC vendors. Three main phases have been identified to move from these heterogeneous natural language description of the expected CBTC features to the actual CBTC product requirements. Furthermore, one additional phase is required to define the requirements of the single systems that compose the CBTC product.

Figure 2 summarizes the approach followed. Activities are depicted as circles and artifacts are depicted as rectangles with a wave on the bottom side.

First, domain analysis is performed (Sect. 3). During this phase, the requirements standards are analysed together with the documents of the different vendors. The former are used to identify the functionalities expected from a standard-compliant CBTC system (Functionality Identification), while the latter are used to identify the system architectures adopted by the vendors (Architecture Identification). Requirements standards are also employed in the Architecture Identification task to provide a common vocabulary to describe the architectures.

In the second phase, a product family for CBTC systems is defined (Sect. 4). The architectures identified in the previous phase are evaluated, and a feature model is derived to hierarchically capture all the different architectural options available in the market (Feature Modelling).

The third phase drives the definition of the actual product features (Sect. 5). From the feature model that represents the product family, a product instance is chosen. A detailed architecture is defined for such a product instance, taking into account the functionalities extracted from the standards (Product Architecture Modelling). Then, scenarios are derived to analyse the different behavioural aspects of the product (Product Scenario Modelling).

The final product requirements are the results of the adaptation of the standard CBTC requirements to the desired product. This adaptation is provided according to (1) the functionalities extracted from the standards, (2) the product architecture, and (3) the product scenarios.

In the fourth phase requirements are defined for the individual sub-systems that compose the overall CBTC product (Sect. 6). This phase is oriented to accomplish the process-level requirements prescribed by the CENELEC norms [11,10,12]. First, a Preliminary System Specification (PSS) document is defined, which is based on the functionalities extracted from the product standards and on the product architecture chosen (PSS Definition). Then, an approach based on prototyping is employed to define the System Requirements Specification (SYS-RS) document, which collects the requirements for the system (SYS-RS Definition).

Currently, most of the tasks of the approach are based on engineering activities with limited automation. Such activities have been mainly performed using Microsoft Word[1] and Microsoft Excel[2] documents. Microsoft Visio[3] was employed whenever graphical diagrams were re-

[1] http://office.microsoft.com/en-us/word/
[2] http://office.microsoft.com/en-us/excel/
[3] http://office.microsoft.com/en-us/visio/

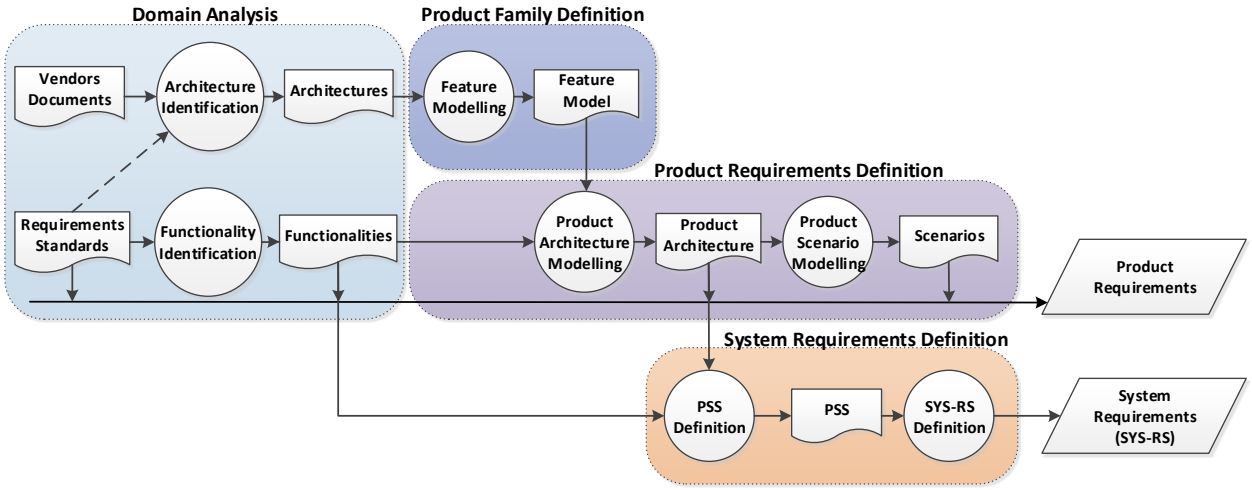


Fig. 2: Overview of the product requirements definition process adopted

quired (i.e., during Architecture Identification, Feature Modelling, Product Architecture Modelling and Product Scenario Modelling). Furthermore, Microsoft Visual Studio[4] was used to implement a prototype system during the System Requirements Definition phase.

Other internal tools, developed by ISTI-CNR, have been also employed in the process. NLP tools for term identification have been experimentally used to identify the features from the Vendors Documents [27], while the QuARS tool [18] was used to detect ambiguities in the SYS-RS document. When appropriate, alternative software packages, and possible tool choices to improve the robustness of the approach, are referred throughout the paper.

3 Domain Analysis

The Domain Analysis phase is composed of two subphases, namely Functionality Identification and Architecture Identification. In the first phase, the available CBTC standards are analysed, and a list of functionalities for CBTC systems is provided. In the second phase, the publicly available documents of the selected vendors are inspected to identify the CBTC architectures available in the market.

3.1 Functionality Identification

In this phase, functionalities are identified for a generic CBTC system by evaluating the available international standards. Currently, the reference standards are IEEE 1474.1-2004 [31] and IEC 62290 [1,2], which are briefly summarized below.

3.1.1 IEEE 1474.1-2004

The IEEE 1474.1-2004 has been defined by the Communications-based Train Control Working Group of IEEE (Institute of Electrical and Electronic Engineers) and approved in 2004. Such standard concerns the functional and performance requirements that a CBTC system shall implement. The requirements concern the functions of Automatic Train Protection (ATP), Automatic Train Operation (ATO) and Automatic Train Supervision (ATS), implemented by the wayside and onboard CBTC system. The ATO and ATS functions are considered optional by the standard. In addition to these requirements, the standard also establishes the headway criteria, system safety criteria and system availability criteria applicable to different transit applications, including the Automated People Movers (APM).

3.1.2 IEC 62290

The IEC 62290 is a standard defined by the IEC (International Electrotechnical Commission) come into effect in 2007. This standard brings the fundamental concepts, the general requirements and a description of the functional requirements that the command and control systems in the field of urban guided transport, like the CBTC, shall possess. In reference to the fundamental concepts, the standard establishes four levels or Grades of Automation (GoA-1 to 4). The increasing GoA corresponds to increasing responsibility of the command and

[4] `http://www.microsoft.com/visualstudio/eng/visual-studio-2013`

control system w.r.t. the operational staff. For example, a GoA-1 system simply enforces brakes when the driver violates the braking curve. A GoA-4 system does not have a driver, nor yet an onboard human supervisor.

The standards have been evaluated to derive a complete set of CBTC functionalities. The approach adopted is as follows. First, the functionalities that the IEEE 1474.1-2004 standard specifies have been extracted. Such functionalities have been divided between ATP, ATO and ATS according to the anticipated classification provided by the same standard. Starting from this first group of functionalities, the activity continued with the analysis of the IEC 62290 standard, for identifying possible additional functionalities in comparison to those already extracted. Each functionality is traced to the paragraph of the corresponding standard from which it has been originally derived. We have derived 67 functionalities in total (see Sect 7 for further details), which have been validated by our industrial partner.

Example functionalities, which are useful to understand the examples reported in the rest of the paper, are reported below together with the related subsystem and the reference to the standard documents.

Train Location Determination. (ATP onboard - IEEE 6.1.1) This functionality determines the position of the train;

Safe Train Separation. (ATP onboard - IEEE 6.1.2) This functionality uses the location information of the train to compute the braking curve and ensure safe separation of trains;

Movement Authority Determination. (ATP wayside - IEC 5.1.4.1) This functionality computes the MA message to be sent to the train based on the position of the other trains and on the railway status;

Route Interlocking Controller. (ATP wayside - IEEE 6.1.11) This functionality controls an external IXL and performs the route requests and locks. IXL systems are normally based on fixed block principles. This function is able to bypass the interlocking inputs concerning the position of the trains coming from the track circuits. In this way, the functionality is also able to ensure the increased performance guaranteed by the moving block principles;

Train Routing. (ATS - IEEE 6.3.4) This functionality allows setting the route for the train in accordance with the train service data, predefined routing rules and possible restrictions to the movement of the train;

Train Identification and Tracking. (ATS - IEEE 6.3.3) This functionality monitors the position and the identity of the trains.

ATS User Interface. (ATS - IEEE 6.3.2) This functionality implements the graphical user interfaces to display the status of the metro, and to allow the operator to perform supervision of the overall system.

3.2 Architecture Identification

In this phase, different possible architectures for a CBTC system are identified by evaluating the available information about the CBTC products on the market.

Several implementations of CBTC systems are offered by different vendors. In our work, we focused on the systems proposed by Bombardier [46], Alstom [45], Thales [50], Invensys Rail Group [32], Ansaldo STS [3], Siemens [44], and GE Transportation [28].

The major subsystems identified in the evaluated CBTC systems are ATP, ATS, ATO and IXL. The adopted terminology is the one provided by the CBTC standards, since the vendors use slightly different terms to refer to the same components. There are also other additional subsystems, which include, e.g., the fire emergency system, the passenger information system, and the closed-circuit television.

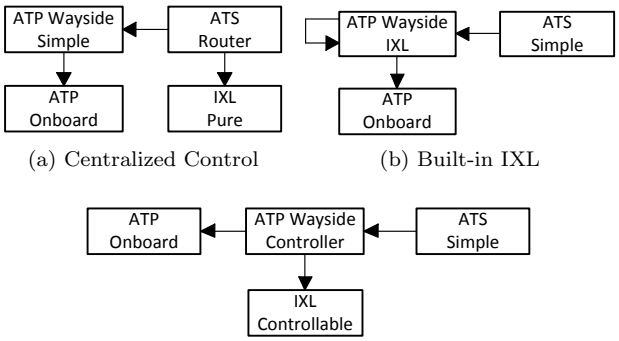
The possible CBTC architectures have been identified by analyzing the relationship between the different subsystems. As examples, we focus on the relationships among ATP, ATS and IXL. The most relevant configurations identified for these systems are summarized below.

Centralized Control. (Figure 3a) In this configuration, the ATS controls both the ATP and the IXL. The ATS is called `ATS Router` since it has a direct interface with the IXL to perform routing. The wayside ATP is called `Wayside ATP Simple` since it has no direct interface with the IXL, and the communication among these two subsystems is managed through the ATS. Furthermore, the wayside ATP communicates with the onboard ATP, as in all the other configurations.

Built-in IXL. (Figure 3b) In this configuration there is no external IXL, since the ATP encapsulates also the functions of the IXL (`ATP Wayside IXL`). We call the ATS of this configuration `ATS Simple` since it has no direct interface with an IXL.

Controllable IXL. (Figure 3c) The wayside ATP has a control interface (`ATP Wayside Controller`) with an external IXL, and acts as intermediary between the `ATS Simple` and the IXL. We call the IXL of this configuration `IXL Controllable` since, unlike the `IXL Pure` of the first configuration, allows the `ATP Wayside Controller` to bypass some of its controls to achieve improved performances. It is worth noting that this solution would not be possible with an ATS controlling the IXL. Indeed, the ATS is normally not meant as a safety-related system, while the ATP and the IXL are safety-critical platforms.

Configurations 3a and 3b are both used by Bombardier. The second architecture is described in the Bombardier documentation as CITYFLO 650 with built-in IXL. Though architecture 3a is not explicitely described, the Bombardier documentation states that, when available, the IXL works as a backup system in case of ATP failure.



(a) Centralized Control (b) Built-in IXL

(c) Controllable IXL

Fig. 3: Architectures extracted

Therefore, we can argue that the IXL control resides in the ATS and not in the ATP.

Architecture 3c has been derived evaluating the Alstom system. The IXL employed by Alstom is provided by the same supplier of the Bombardier IXL, but Alstom does not use this IXL as a backup system. Therefore, we can argue that the ATP is in charge of controlling the IXL, as in architecture 3c. Though this type of architecture really complicates the safety-case, it is the only way to achieve the benefits of the moving block principles in an area that is controlled by an IXL.

4 Product Family Definition

The development of industrial software systems may often profit from the adoption of a development process based on the so-called *product families* or *product line* approach [19,14]. This development cycle aims at lowering the development costs by sharing an overall reference architecture for each product. Each product can employ a subset of the characteristics of the reference architecture in order to, e.g., serve different client or jurisdictions.

The production process in product lines is hence organized with the purpose of maximizing the commonalities of the product line and minimizing the cost of variations [39]. A description of a product family (PF) is usually composed of two parts. The first part, called *constant*, describes aspects common to all products of the family. The second part, called *variable*, represents those aspects, called variabilities, that will be used to differentiate a product from another. Variability modelling defines which features or components of a system are optional, alternative, or mandatory.

The product family engineering paradigm is composed of two processes: *domain engineering* and *application engineering*. *Domain engineering* is the process in which the commonality and the variability of the product family are identified and modelled. *Application engineering* is the process in which the applications of the product family are built by reusing domain artefact and exploiting the product family variability [39].

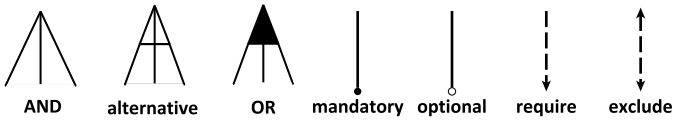


Fig. 4: Feature diagram notations

4.1 Feature Modelling

The modelling of variability has been extensively studied in the literature, with particular focus on *feature modelling* [34,4,15]. Feature modelling is an important technique for modelling the product family during the domain engineering.

The product family is represented in the form of a *feature model*. A feature model is as a hierarchical set of features, and relationships among features. A formal semantics is defined for these models, and each feature model can be characterized by a propositional logic formula [4,41].

Relationships between a parent feature and its child features (or subfeatures) are categorized as: *AND* - all subfeatures must be selected; *alternative* - only one subfeature can be selected; *OR* - one or more can be selected; *mandatory* - features that are required; *optional* - features that are optional; *a require b*, if the presence of *a* requires the presence of *b* ; *a exclude b*, if the presence of *a* excludes the presence of *b* and vice-versa.

A *feature diagram* is a graphical representation of a feature model [34]. It is a tree where primitive features are leaves and compound features are internal nodes. Common graphical notations are depicted in Figure 4.

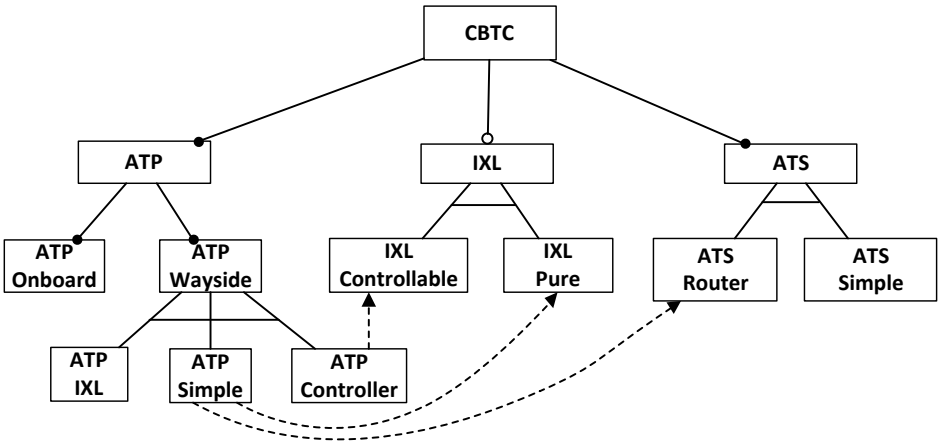


Fig. 5: Simplified excerpt of the CBTC global feature diagram

4.2 A Global Feature Diagram for CBTC

A global feature model for CBTC has been defined by integrating the different architectural choices identified during the architecture identification task (Sect. 3.2). We show the model for the GoA-1 level, according to the IEC 62290 terminology [1]. In other terms, such a model assumes the presence of a driver on board (i.e., there is no ATO system). In our current experience, we have defined a global feature model that includes the ATO as well. However, for space and clarity reasons, we discuss the model only for the GoA-1 level.

An informal bottom-up approach has been followed to pass from the architectures to the global feature diagram. First, all the identified components have been considered as *leaves* of the diagram. Then, internal nodes and hierarchy are provided for those components that occurred with different variants in the architectures. Finally, constraints are provided by inspecting the different architectures: if a component always occurs together with another component, a *require* constraint is defined.

A simplified excerpt of the global feature diagram associated to our model is given in Figure 5. The diagram includes the architectural components (which in our diagram become *features*) already identified in Sect. 3.2.

The *require* constraint requires a product to include `IXL Pure` and `ATS Router` whenever the product includes `ATP Simple`. Indeed, the control interface with the IXL has to be implemented by the ATS if the ATP does not include it, as in the case of `ATP Simple`. Also `IXL Controllable` is required whenever the `ATP Controller` is used. In this case, a proper controllable interface of the IXL is required to let the ATP system control its functionalities.

The `ATP Onboard` is required by any product of this family. On the other hand, the features `IXL Pure` and `IXL Controllable` cannot cohabit in any product of this family. The same observation holds for `ATS Router` and `ATP Simple`. Indeed, only one type of IXL and one type of ATS is allowed in a product.
It is worth noting that the feature diagram allows new configurations that were not identified in the domain analysis phase performed. These configurations represent new possible products. For example, an `ATP IXL` can - optionally - cohabit with an `IXL` of any type. In this case, the additional `IXL` works as a backup system.

The propositional logic formula associated to the excerpt is the conjunction of the formula of the `ATP` subtree with the formulas associated to the `IXL` and `ATS` sub-trees, and with the *require* constraints. For example, the formula associated to the `IXL` sub-tree is:

$$(\texttt{CBTC} \wedge true) \wedge (\texttt{IXL} \Rightarrow \texttt{CBTC}) \wedge ((\texttt{IXL Controllable} \Leftrightarrow (\texttt{IXL} \wedge \neg \texttt{IXL Pure})) \wedge (\texttt{IXL Pure} \Rightarrow (\texttt{IXL} \wedge \neg \texttt{IXL Controllable})))$$

Similar formulas can be written for the other sub-trees and for the *require* constraints. Tools such as Splot [36], can be used to verify the consistency of the feature model, and check for the presence of dead features (i. e., features that cannot be instantiated in any product), or inconsistent relationships among features.

5 Product Features Definition

The provided feature model represents a global model for CBTC at the GoA-1 level. From this global model we choose a product instance, which in our example case corresponds to the Controllable IXL architecture of Figure 3c. Then, we model the *detailed architecture* of the product according to the functionalities extracted from the standards in the domain analysis phase. The architecture represents a static view of our product in the form of a block diagram. In order to assess the architecture, we provide realistic scenarios using architecture-level sequence diagrams. This phase can be regarded as the application engineering process of the product fam-

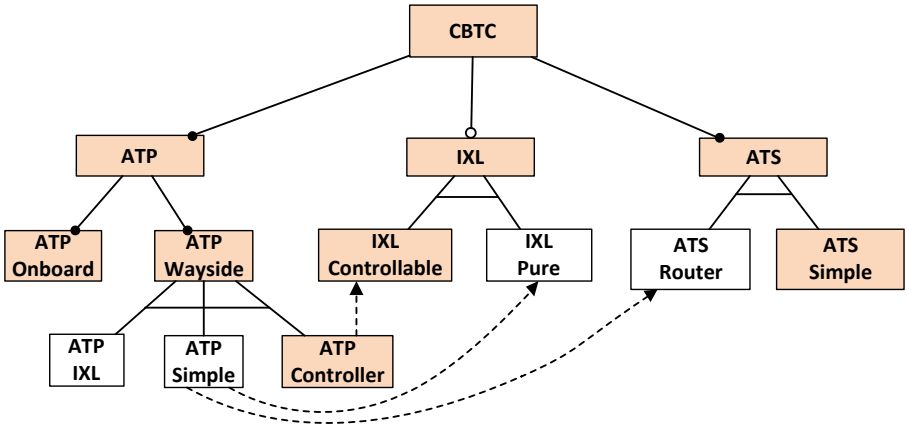


Fig. 6: Selection of features for our example product

ily engineering paradigm. Architecture and scenarios are employed to derive requirements for the actual product.

5.1 Product Architecture Modelling

The graphical formalism adopted to model the product architecture is a block diagram with a limited number of operators. We have designed this simple language according to our previous experiences in the railway industry [24,23]. Companies tend to be skeptical about the benefit given by the adoption of complex and rigid languages during the early stages of the development. Instead, they are more keen to accept a lightweight formalism that allows them to represent architectures intuitively and with a limited effort.

The diagrams are composed of blocks and arrows. Blocks can be of two types: *system blocks*, which represent individual hardware/software systems, or *functionality blocks*, which represent hardware/software functionalities inside a system. Two types of arrows are also provided: *usage arrows*, allowed between any block, and *message arrows*, allowed solely between functionalities belonging to different systems. If a usage arrow is directed from a block to another, this implies that the former uses a service of the latter. If a message arrow is directed from a functionality to another, this implies that the former sends a message – the label of the arrow – to the latter.

We describe the usage of this formalism with an example. Given the global CBTC model, we first select the features that we wish to implement in our final product. For example, Figure 6 highlights in pink (grey if printed in B/W) the features that are selected for a CBTC system that uses a controllable interlocking (see Figure 3c).

An excerpt of the detailed architecture for the selected product is depicted in Figure 7. It is worth noting that the functionality blocks used are part of the functionalities identified during the domain analysis phase. The selection and apportionment of such functionalities is manually performed by the person who defines the detailed architecture.

The `Train Location Determination` functionality belonging to the onboard ATP sends the train location information to the ATP wayside system. The `Movement Authority (MA) Determination` functionality forwards this information to the ATS for train supervision, and uses this information to compute the MA. The MA is sent to the `ATP Onboard` – to enforce train separation – and to the `ATS User Interface`, which visualizes the MA. The `Train Routing` functionality of the ATS requires the routes to the wayside ATP, which controls the routing by means of the `Route Interlocking Controller` functionality connected to the IXL. We recall that the `Route Interlocking Controller` functionality is used to modify the interlocking inputs concerning the location of the trains – normally based on fixed block principles – to achieve the increased performance of the moving block paradigm.

5.2 Product Scenario Modelling

The architecture provided during the previous activity has been defined according to the functionalities extracted from the standards. Nevertheless, some connections among functionalities, or some message exchange, might be missing from the model, since the architecture has not been evaluated against actual scenarios. In order to refine the architecture, and provide coherent requirements for the product, graphical scenarios are defined.

The graphical formalism adopted to model the scenarios at the architectural level is a modified version of the UML sequence diagrams. Lifelines are associated to systems, while blocks along the lifelines are associated to the functionalities of the system. The arrows among different blocks indicate message communication or service requests. In case of message communication, the arrow is dashed. In case of service requests the arrow is solid. We argue that the proposed notation can be regarded as a

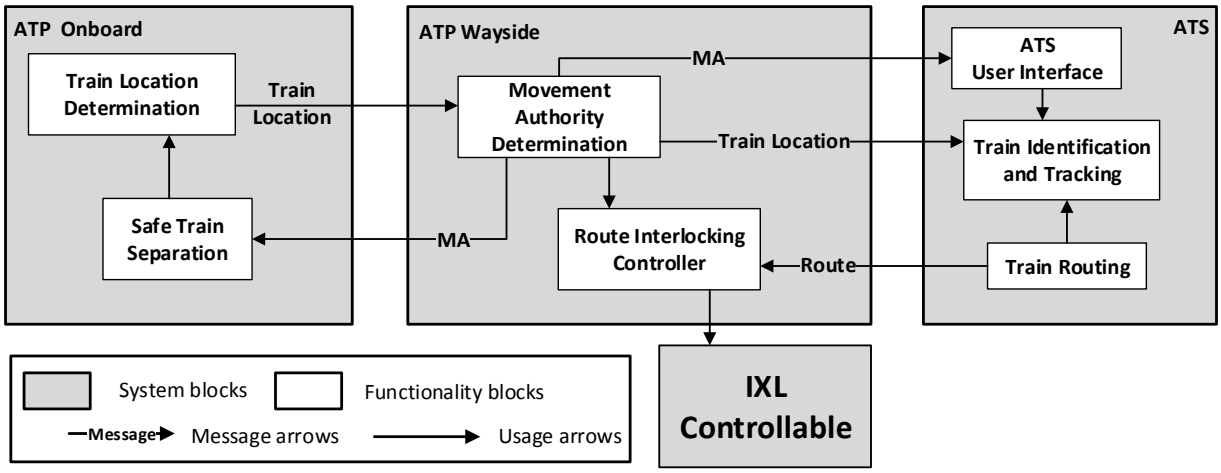


Fig. 7: Architecture example for a CBTC system

high-level sequence diagram notation. Indeed, it is simpler than UML sequence diagrams, but it has the proper level of abstraction for the system definition phase, while UML sequence diagrams are more suitable for the software design phase. Furthermore, fuctionalities are displayed along the lifelines of the systems: this is normally not possible with UML sequence diagrams.

Figure 8 reports a scenario for a train that moves from a station to another according to a route defined by the ATS.

In the operational center, the ATS sends the `Route` information to the wayside ATP. The wayside ATP requests the IXL to move the switches in the proper position, and to lock the resources (the `setRoute` service request). Once the route has been locked by the IXL (`LockEvent`), the wayside ATP sends the `Movement Authority` to the onboard ATP for a first train (`ATP Onboard (T1)`) and to the ATS, which displays the MA. The onboard ATP allows the train departure, so the driver can start the train movement. While moving, the onboard system updates its position and sends the `Train Location` information to the wayside ATP. This system uses such information to compute new MAs for the current and preceding trains (represented by `ATP Onboard (T2)`). Furthermore, the wayside ATP forwards the `Train Location` information to the ATS for identification and tracking.

It is worth noting that, in this representation, we have added the `setroute` service request and the `LockEvent` message, which were not defined in the block diagram. The explicit request, and the corresponding response, are an example of refinement enabled by the usage of scenarios: the relationship among the `Route Interlocking Controller` functionality and the `IXL Controllable` system has been clarified by means of the sequence diagram.

5.3 Requirements Definition

The information provided throughout the process are used to define the requirements of the final product. In particular, the requirements of one of the standards are used as a reference for the definition of the actual product requirements. In our case, we take the IEEE 1474.1-2004 standard as a reference.

The requirements are tailored according to the functionalities extracted from the standards, and evaluating the product architecture and the scenarios. For example, consider the following requirement referred to the ATP system:

6.1.11 – **Route Interlocking.** A CBTC system shall provide route interlocking functions equivalent to conventional interlocking practice to prevent train collisions and derailments. [...]
Where an auxiliary wayside system is specified by the authority having jurisdiction, interlocking functions *may* be provided by separate interlocking equipment [...].

In our example product, the interlocking is an auxiliary wayside system, external to the ATP. Therefore the Derived (D) requirement for our product is:

6.1.11(D) – **Route Interlocking.** Interlocking functions *shall* be provided by separate interlocking equipment [...].

Additional requirements on the actual behaviour can be derived from the architecture and the example scenario, as in the following:

6.1.11($D-1$) – **Route Interlocking Controller.** When a route is requested from the ATS, The ATP system shall require route setting (`setRoute`) to the interlocking to lock the interlocking resources. [...]

The behaviour expected from this requirement is clarified by the scenario, which is also attached to the requirement in the final specification. At this stage, we

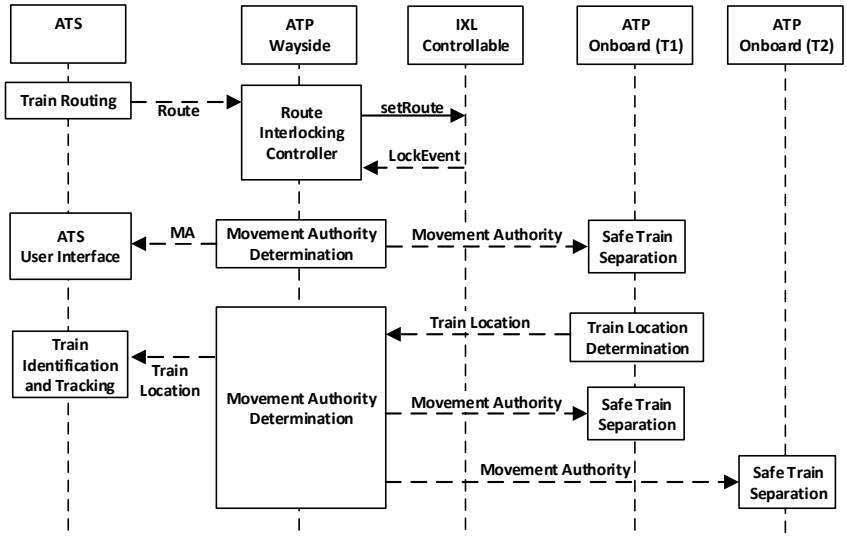


Fig. 8: Example sequence diagram: a train moves from one station to another

did not find general patterns for passing from the standard requirements to the product requirements. Indeed, the definition of the requirements is a manual process, where each requirement of the standard is reviewed and properly extended/reduced according to the results of the previous phases.

Consider now a vendor that wishes to accomplish also the IEC 62290 standard with his product. The product is already defined according to IEEE 1474.1-2004 following the presented approach. In this case, we argue that the compliance with the IEC 62290 standard can be demonstrated by reasoning at functional level. Indeed, the functions identified in the domain analysis phase integrate the content of both standards, and traceability with the original functional requirements of IEC 62290 is therefore made easier.

6 System Requirements Definition

The development of railway and metro signalling platforms in Europe shall comply with the CENELEC standards [12,11,10]. These are a set of norms and methods to be used while implementing a product having a determined safety-critical nature. If a company wishes to achieve a CENELEC certification for its CBTC product, the development of the product shall follow the guidelines and the prescriptions of the norms. In principle, the company can decide to treat the CBTC product as a single system, and provide certification for the system as a whole. Nevertheless, once the company has to sell a product variant, the certification process shall be entirely performed also for the variant, paying undesirable costs in terms of budget and time.

Therefore, it is useful to develop each sub-system as an independent unit, and follow the CENELEC regulations for the development of such sub-system. Once each sub-system has got certification evidence according to the regulations, the certification of the whole CBTC product is made easier, since it can be focused solely on the integration aspects. Furthermore, if the customer requires only a specific sub-system (e.g., the ATP or the ATS system) to renew a part of its installation, the subsystem can be purchased without additional certification costs. The first documents typically edited for the development of a system in a CENELEC-compliant process are the Preliminary System Specification (PSS) and the System Requirements Specification (SYS-RS). The former is a document that summarizes the interfaces of the system, and the functionalities that are expected from the system. The latter is a document that precisely specifies the expected system behaviour, as well as the safety, performance, architectural and environmental constraints. Both documents are normally written in natural language.

In our approach we suggest to derive the PSS directly from the detailed architecture. Moreover, we apply scenario-based requirements elicitation [47], aided with rapid prototyping [29] to produce the SYS-RS document. Moreover, our method expects the SYS-SRS document to be produced in a constrained natural language.

6.1 PSS Definition

The approach for the definition of the PSS is as follows. First, we select from the product architecture the subsystem to be developed. We choose, for example, the `ATS Simple` introduced in Sect. 3.2 and employed in the example of Sect. 5. The information provided by the detailed architecture diagram for the ATS is the same information required by the PSS document. The *message*

ID	Type	Data	From	To
E.01	WLAN	Train Location	ATP	ATS
E.02	WLAN	Route	ATS	ATP
E.03	TD	MA	ATS	Operator

Table 1: Excerpt of the interfaces of the ATS sub-system

arrows are the interfaces, while the *functionality blocks* are the expected functionalities. Therefore, the definition of such a document comes straightforwardly from the detailed architecture diagram.

In Table 1, we give an excerpt of the PSS of the ATS sub-system concerning the interfaces with the other subsystems or actors.

We notice that the table includes also the *type* of interface. Indeed, design decisions concerning the types of interfaces and the types of the devices shall be provided in the current phase. In particular, we see that the Movement Authority (MA) is displayed to the user through the Train Describer (TD), which is a screen that displays the metro layout and the information concerning the position and the MAs of the trains (an example TD can is reported in Fig. 10).

While the functionalities are extracted from the detailed architecture diagram, the natural language details concerning such functionalities can be directly extracted from the Domain Analysis Phase (see Sect. 3.1). However, in some cases, the details provided might not be sufficient to precisely specify the functionalites of the system. Moreover, the PSS document shall take into account the design decisions taken. For example, the ATS User Interface extracted from the standards does not give details concerning the devices for the visualization of the information concerning the metro status. In these cases, sub-functionality partitioning is required. Below, we give an excerpt of PSS of the ATS concerning the partitioning of the ATS User Interface (in our PSS document, functionality F4), with focus on the sub-functionality F4.1 related to the already mentioned Train Describer.

F4: ATS User Interface (IEEE 6.3.2) This function implements the visualization of all the information that are required for the monitoring and the management of the CBTC system. [...]

- F4.1. Management of the Train Describer: This function provides real-time information concerning the status of the metro network. It is a view of the system containing:
  - a scaled representation of the metro layout;
  - the position of the trains in real-time. Each train is identified by a unique number;
  - information concerning the busy routes and the free routes, highlighted in different colors;
  - information concerning the Movement Authority (MA) of each train.
- F4.2. Management of the Train Graph [...]
- F4.3. Provide interface to the operation control centre HMI (IEC 6.2.2.5.1)) [...]
- F4.4. Provide interface to the decentralized HMI (IEC 6.2.2.5.2) [...]

We notice that, for those functionalities that have been extracted from the CBTC standards, the reference to the original standard is reported in the PSS. For the additional functionalities required by the design decisions, and therefore not strictly related to the standards, the reference cannot be provided. Nevertheless, we have experienced that the number of such functionalities is quite limited. Furthermore, in most of the cases, these additional functionalities are sub-functionalities of those expressed in the standards, as in the presented example.

6.2 SYS-RS Definition

The System Requirements Specification (SYS-RS) is the main reference document, which is used in the subsequent process phases for both the development and the system verification. Requirements in the SYS-RS document are normally partitioned into technological, interface, functional, performance, RAM - Reliability, Availability, Maintainability - and safety requirements. Here, we focus on the definition of *interface* and *functional* requirements. Requirements are normally written in natural language, and, following the CENELEC norms, they shall be *complete*, *clear*, *precise*, *unequivocal*, *feasible*, *verifiable*, *testable* and *maintainable* [11]. Here, we focus on the first five attributes.

In our approach, we employ a scenario-based iterative approach aided with prototyping for requirements definition. Such an approach enforces requirements completeness and feasibility. Furthermore, requirements are written in a constrained natural language. This choice enforces the production of clear and precise requirements. Requirements are also analysed through the QuARS tool for natural language ambiguity detection [18], in order to produce unequivocal requirements.

Figure 9 illustrates the approach. First, functionalities are selected from the PSS. For each functionality, we elicit one or more behavioural scenarios in the form of natural language stories. From each scenario we derive requirements in a constrained natural language. Such requirements are analysed by means of the QuARS tool. Once all the functionalities have been evaluated and the scenarios have been written, the requirements are implemented in an executable prototype. The executable prototype is used to derive new possible scenarios, and therefore new requirements. The approach iterates until no additional scenario is foreseen.

Scenario Definition The approach is as follows. First, we derive natural language scenarios starting from the functionality listed in the PSS document. Scenarios are

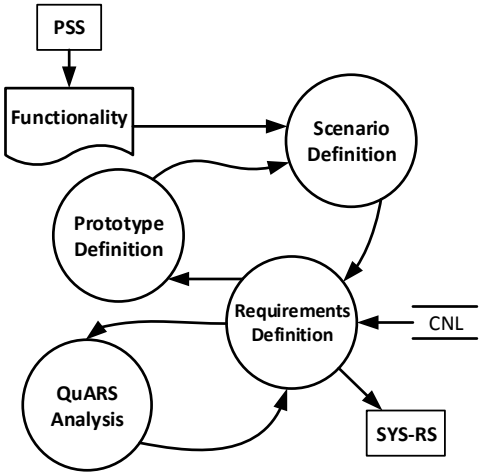


Fig. 9: Approach for the definition of the SYS-RS

ID: 001	**TITLE:** Visualization of the Movement Authority
SOURCE: F4.1 Management of the Train Describer	
1. The ATS receives a message from the ATP	
2. The ATS unpacks the message and recognizes that it is a message of type Movement Authority - MA	
3. The ATS realizes that the MA contained in the message is associated to the train numbered T	
4. The ATS visualizes the length of the MA through the user panel in the point of the railway yard where the train numbered T is currently placed	

Table 2: Example scenario derived from the functionality F4.1 *Management of the Train Describer*

written in the form of bullet-list stories. This approach enables the elicitation of possible system usages, while we employ natural language - and not, e.g., sequence diagrams - in order to involve the largest amount of stakeholders in the scenario definition. Indeed, we argue that a restricted UML language, such as the one presented in Sect. 5.2, is normally understandable by all the stakeholders, but cannot be profitably used by all of them to design scenarios during the requirements elicitation phase. With natural language scenarios, we can ask the largest amount of stakeholders to write the scenarios and explore possible system's usage.

The format of the scenario shall follow a few simple rules. Each scenario shall have an *identifier*, a *title* in natural language, a *source* functionality, and a *list of actions* that describe the scenario. In Table 2, we report one of the scenarios that have been derived from the functionality F4.1. *Management of the Train Describer*.

Requirements Definition From each scenario, we derive a set of natural language requirements in a constrained natural language. Several types of constrained natural languages (CNL) have been proposed in the literature (see [43,30] for some examples, and [7] for a list of domain specific CNLs). However, all such languages had limited use in practice, since they often appear as too complex to handle, and too complex to be read.

In our approach, we use a constrained natural language that is inspired to the language succesfully employed in the MODCONTROL project [9]. The format is based upon four simple formats that shall be employed to write requirements. The formats are reported below:

FORMAT1. The system [*shall*|*should*] be able to $<$ *capability* $>$.

This format is employed in case of requirements that involve mandatory (shall) or optional (should) functionalities, which are *unconditional* and *independent* from the actions of the operators. Requirements of this type are normally associated to interface functions, internal procedures, or procedures that manage internal data structures.

FORMAT2. The system [*shall*|*should*] allow the $<$ *operator* $>$ *to* $<$ *action* $>$.

This format is employed in case of requirements that involve mandatory (shall) or optional (should) functionalities, which are *unconditional* and *dependent* from the actions of the operators. A requirement of this type is *"The system shall allow the supervising operator to select the train to stop at the next station"*.

FORMAT3. The system [*shall*|*should*] $<$ *action* $>$, [when|after|before|if] $<$ *condition* $>$ {, [when|after|before|if] $<$ *condition* $>$}.

This format is employed in case of requirements that involve mandatory (shall) or optional (should) system actions that depend on one or more conditions. All conditions are considered in a logical AND relationship. If we want to express logical OR among conditions, it is recommended to add a new requirement.

FORMAT4. [FORMAT1|FORMAT2|FORMAT3], $<$ *procedure* $>$.

The format is a combination of one of the previous formats with a procedure. This format is employed in case of requirements that involve functionalities that have an associated procedure, or that are performed through a well-defined interface device. The format shall be used when it is useful to explain *how* the system is expected to perform a certain action.

The fields $<$ *capability* $>$, $<$ *action* $>$, $<$ *condition* $>$ and $<$ *procedure* $>$ are free-form sentences, with the only constraint of containing one verb maximum.

Below, we report the requirements that have been derived from the scenario of the previous paragraph, together with the format of the requirement (FORM-N = FORMAT N).

1. The system shall be able to receive messages from the ATP system (FORM-1);

2. The system shall parse the message, when the system receives a message (FORM-3);
3. The system shall identify the type of the message, after the system has parsed the message (FORM-3);
4. The system shall identify the fields of the message, after the system has identified the type of the message (FORM-3);
5. The system shall display the length of the Movement Authority (MA) of a train T, when the system receives a message of type MA with field TRAIN_ID = T (FORM-3);
6. The system shall display the length of the MA of a train T, through the Train Describer (FORM-4);
7. The system shall display the length of the MA associated to a train T, in the point of the railway yard where the train T is currently placed (FORM-4);

In this example, we do not have requirements of FORMAT 2, since the Train Describer does not allow interaction with the operator.

After the definition of the requirements, these are partitioned into *functional* and *interface* requirements. For example, requirement 1, 6 and 7 will be part of the *interface* requirements. All the other requirements can be considered *functional* requirements.

We argue that the proposed constrained natural language has several advantages in the considered domain. It is easy to use, since the formats can be easily remembered. It is sufficiently strict to highlight the relevant capabilities, actions, conditions and procedures. Therefore, it enables the production of *precise* requirements. Furthermore, it naturally produces short sentences, since only one verb is admitted in the free-form fields, and this enables the production of *clear* requirements.

Requirements shall be *unequivocal*, according to the CENELEC norms. In order to enforce this quality attribute, we employ the QuARS tool for requirements analysis. The tool detects potential natural language ambiguities in the requirements by searching for typically ambiguous expressions. For example, the terms "clear", "easy", "adequate" indicate *vagueness*, the expression "as < *adjective* > as possible" indicate *subjectivity*, and demonstrative adjectives ("this", "that") or personal pronouns ("it", "they") often reveal the presence of an *implicit* - and therefore possibly ambiguous - subject in the sentence. Requirements such as "The system shall handle incoming messages as rapidly as possible" or "The system shall display the position of a train, if it is active" (is "it" referred to the train or to the system?), are identified as ambiguous by QuARS. Such requirements shall be rephrased, modified, or removed after the QuARS analysis. To have a complete view of all the types of ambiguities that the tool identifies, please refer to [18]. The current capabilities of QuARS - as well as the capability of similar tools, such as Requirements Assistant[5] - do not go beyond the so-called lexical and syntactic ambiguities. Works are currently ongoing to discover semantic and pragmatic ambiguities [25].

We have experienced that the proposed language is sufficiently flexible to allow the expression of all the *interface* and *functional* requirements required by our context. The other types of requirements (i.e., technological, performance, RAM, safety), normally included in the SYS-RS, may require different formats - they often include numerical constraints - and other derivation strategies (e.g., quantitative models of the system).

Furthermore, we argue that the presented CNL is a starting point towards a formal representation of the requirements. For example, requirements 2 can be represented with the following SOCL formula [20]:

$$AG([\text{reveived\_msg}(\$m)](AX\{\text{parse\_msg\_begin}(\%m)\}true)).$$

The formula states that, whenever a message m is received ([reveived_msg(m)]), the parsing procedure for the message m shall start at the next (operator X) system execution step. We notice that the formula includes a parameter (i.e., the message m). Currently, the only tool that supports the SOCL logic with parametric formulas is UMC [49], which is also available on-line[6]. Other experiences have been presented in the literature that aims at transforming natural language requirements into formal specifications (e.g., [37,21]). We argue that the definition of a CNL such as the one presented, can be a proper intermediate step to achieve this goal. Indeed, having a reduced amount of formats can help identifying those fragments that can be transformed into logic formulas - and verified through model checking - and those that do not have a corresponding logic representation - and need to be verified through model/code inspection or testing.

Prototyping The scenarios can be regarded as a starting point for requirements elicitation, but they are not sufficient to enforce the *completeness* of the requirements, required by the norms. Several possible system usage and features might be missing. Therefore, in order to enforce requirements completeness, the requirements that are derived from the scenarios are implemented in a prototype. The prototype might be implemented either in a programming language, or with formal/semi-formal modelling. The relevant aspect is that the prototype shall be *executable*. Interaction with the prototype helps deriving new possible usage scenarios to elicit new requirements. The prototype enables the identification of scenarios that can hardly be foreseen if one focuses solely on one functionality, as we do when we derive the first set of scenarios. Indeed, exercising the prototype highlights issues related to the *interaction* among functionalities. Moreover, the prototype helps discovering issues that are related to the implementation, and that shall be considered in the requirements.

[5] http://www.requirementsassistant.nl

[6] http://fmt.isti.cnr.it/umc/V4.1/umc.html

For example, consider the requirements of the previous paragraph. The prototype implements such requirements, as well as all the other requirements derived from the other scenarios. To have an executable prototype that is capable of executing the scenario, we implement the following components:

- a stub function that emulates the communication part of the ATP, and sends the message to the ATS prototype;
- a communication interface that receives the ATP messages;
- a graphical user interface that represents the Train Describer (in Fig. 10, we show the interface of the TD of our prototype).

We execute the original scenario on the prototype, to assess that the provided requirements are sufficient to perform the scenario. Furthermore, we apply some variations to the scenario, exercising the prototype with different, manually defined, input data. For example, we start sending more than one message with the same content, and we see that a policy is required to handle *duplicate messages*. Then, we try to send two messages associated to the same train T, where the MA are *inconsistent* (i.e., they are positioned into different parts of the layout). A policy is required to handle also this situation, since, by default, the graphical user interface of the prototype will show the same MA in different parts of the railway yard. We write down natural language scenarios for these cases, and we derive additional requirements. The additional requirements, in this case, are:

- The system shall discard the message received from the ATP, when the system receives a duplicate message (FORM-2);
- The system shall be able to detect inconsistent MAs (FORM-1);

Furthermore, we require to change requirement 5 of the example as follows:

- The system shall display the length of the Movement Authority (MA) of a train T, when the system receives a message of type MA with field TRAIN_ID = T, if the system did not detect inconsistent MAs (FORM-3).

The concepts of *duplicate message* and *inconsistent MA* are defined in the definition section of the SYS-RS, expressed in free-textual form:

- *duplicate message*: a message that has all the fields equal to the previous message.
- *inconsistent MAs*: an MA is inconsistent with the previous MA, if they are associated to the same train T, if the former starts at M meters, the latter starts at L meters, and $L - M > \tau$.

τ is the tolerance, which is a configuration parameter for the system.

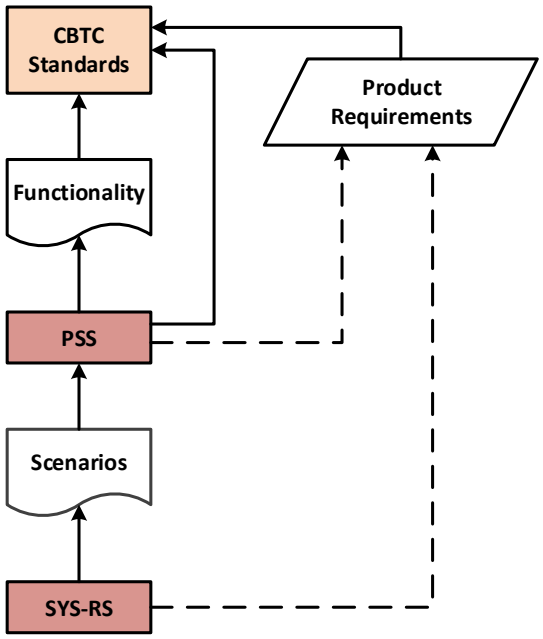


Fig. 11: Traceability links among artifacts in the system definition phase (solid line=explicit link, dashed line= manual link)

The new requirements are implemented in the prototype. Now, the prototype can be exercised again with new scenarios - which are made possible by the extension of the prototype - and new requirements might be issued. In our context, one may require a more fine-grained function that takes into accout the train speed to identify inconsistent MAs.

The choice of stopping the iteration of scenarios-requirements-implementation is up to the team. In our experience, two to three iterations are sufficient to achieve a degree of *completeness* of the requirements that can be acceptable for the team.

Furthermore, since all the requirements are implemented in the prototype, the approach naturally enables the production of *feasible* requirements, as required by the norms.

We argue that a proper way to organize the scenarios shall also be foreseen, in order to guide their navigation, and reason about the interaction among them. We are currently working in this direction.

6.3 Traceability

The CENELEC norms ask for traceability among development artifacts. Moreover, we are here interested also in providing traceability links with respect to the CBTC standards.

Figure 11 depicts the traceability links enforced by our approach. The Functionalities extracted from the

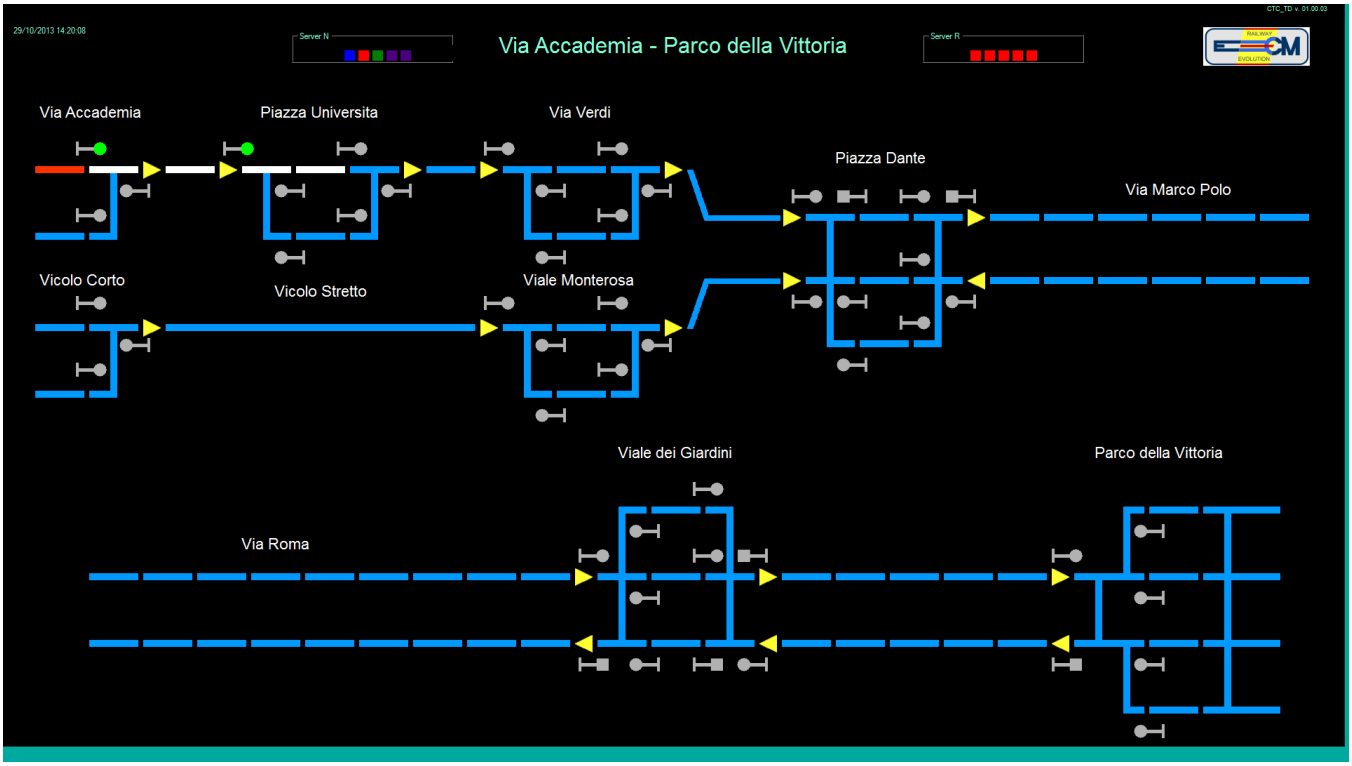


Fig. 10: Interface of the Train Describer of the implemented prototype. The red segment (first segment on the top-left corner) represents the part of the track that is currently occupied by a train. The white segments (second four segments on the top-left corner) represent the Movement Authority associated to the train.

standards are traced back to the source standard. The PSS document is built upon these functionalities and has a direct traceability link to them. The scenarios are derived from the functionalities of the PSS, and the *source* field of each scenario provides the traceability link. Each requirement in the SYS-RS is derived from the scenarios, and therefore each requirement can be traced to the PSS through the scenarios. Traceability links are also reviewed to assess that the additional information added in the SYS-RS - e.g., an additional interface -, is also reported in the PSS. Since many of the steps of the presented process are manual, the validation of the traceability links is important to assess the mutual consistence and quality of the different artifacts.

The link between the Product Requirements and the PSS/SYS-RS is not explicit, and traceability among the artifacts shall be manually performed. However, manual tracing is supported by the link between the Functionalities and the CBTC Standards. Furthermore, manual tracing can help discovering aspects of the CBTC standards - from which the Product Requirements are derived - that have been overseen in the definition of the PSS/SYS-RS. Such manual activity can be regarded as a validation of the compliance of the system documents w.r.t. the CBTC standards.

7 Current Experience

The approach presented in this paper has been defined and experimented in the context of the Trace-IT project, focused on the definition of innovative solutions for intelligent transport systems. The examples reported in the paper are adapted from the deliverables of the project. The project involves two research groups coming from academia (4 people from ISTI-CNR, and 3 people from the University of Florence), and one group coming from a medium-sized railway signalling company (2 people).

The research groups from academia cover the role of technology experts, thanks to the previous experience on product line modelling, and on requirements definition and analysis. The company covers the role of domain expert. The research groups have implemented the process described in the paper, while the company has monitored the activities and has given recommendations and guidelines concerning the domain-related aspects.

The approach has been implemented as follows. The research groups have first analysed the CBTC standards, deriving 67 functionalities (47 from the IEEE standard and 20 from the IEC standard). Table 3 summarizes the number of functionalities associated to each sub-system.

Then, the documents of the vendors have been evaluated and a global feature model was derived, as described in Sect. 4. A product instance has been chosen from the

Source	ATP W.	ATP O.	ATS	ATO
IEEE	15	10	18	4
IEC	2	7	10	1
Total	17	17	28	5

Table 3: Number of functionalities of the standards associated to each sub-system (ATP W. = ATP Wayside, ATP O. = ATP Onboard).

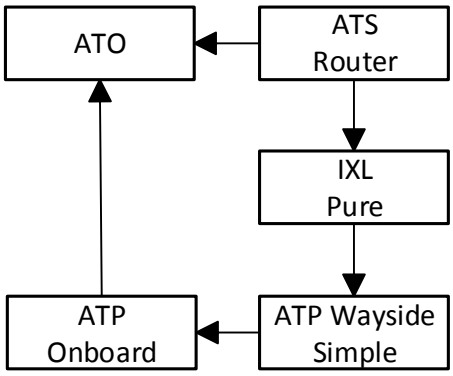


Fig. 12: Architecture of the chosen product instance

diagram. It was taken into account that the company already developed both a CENELEC compliant ATP system, and a CENELEC compliant IXL system (in its IXL pure form). The architecture of the product instance is depicted in Figure 12.

The CBTC system is as follows. The `ATS Router` has a communication link with the `IXL Pure`, and requests routes to such system. The IXL is connected to the `ATP Simple`, since the latter creates MA based on the information concerning the status of the routes that comes from the `IXL Pure`. We notice that the chosen architecture includes also an `ATO` system. The `ATS Router` has a communication link with this system, that is used to send missions (i.e., speed profiles and station stops) to the `ATO` system. The `ATP Onboard` is connected to the `ATO` system. Indeed, the ATO can be regarded as a virtual train driver that shall be controlled by the ATP.

We notice that, regardless of the presence of the ATO system, the presented architecture is completely new with respect to the architectures of the competitors. Indeed, none of the other architectures has a control link between the IXL and the `ATP Wayside` system. Therefore, we have practically seen that the presented approach actually enables the definition of new product architectures that were not available in the market.

System requirements have been defined for the CBTC system according to the approach described in Sect. 5. For confidentiality reasons, the examples reported in this section of the paper do not refer to the actual system requirements for the CBTC product used in the project. However, we argue that such examples are sufficient to clarify the approach. After the definition of the CBTC product requirements, the two research groups operated independently for the development of the CENELEC documents of the ATO (University of Florence group) and for the CENELEC documents of the ATS (ISTI-CNR group). Before the definition of the PSS and the SYS-RS documents, the two groups participated to the definition of the communication protocol between the ATS and the ATO. A communication protocol was required in order to have a clear interface among the systems, to let the two groups work independently.

The ATO group decided to implement a prototype with a semi-formal approach, using IBM Rhapsody[7] as development platform, but *without* implementing the iterative process described in the current paper, and without applying the constrained natural language proposed. Instead the group decided first to write down the requirements in a free-form natural language, and, afterwards, to produce a semi-formal executable model.

The ATS group followed the process described in Sect. 6, and implemented the prototype using the C++ language upon the .NET[8] platform with Microsoft Visual Studio. The choice of following two different approaches was driven by the need to assess the validity of the proposed approach w.r.t. a similar environment where the approach was not applied.

The prototype was developed following the guidelines of the SCRUM development framework [42]. According to the framework, the group performed daily meetings (10 minutes each meeting), where a subset of the requirements was selected and implemented in the prototype.

The ATS group produced 47 scenarios and a SYS-RS composed of 182 requirements and 27 definitions. Two iterations of the approach have been performed to produce the current specification. The part related to the train scheduling functionality, which is part of the ATS, is currently not considered in the specification, since the group decided to perform a separate study for the optimization of the train scheduling.

The current ATO specification includes 51 requirements. Both the ATO and the ATS specifications have been evaluated with the QuARS tool for requirements analysis. The defect rate of the ATS requirements resulted 9% at the first iteration of the approach, and was reduced to 0% in the second iteration. The defect rate of the ATO requirements was 5%. In both cases, the main reasons of the defects was the presence of vague expressions, such as “appropriate”, “imminent” and “shortly before”.

[7] `http://www-03.ibm.com/software/products/us/en/ratirhapfami/`

[8] `http://www.microsoft.com/net`

7.1 Lessons Learnt

Below, we list some lessons that have been learnt during the current experience.

Effort Required During the Domain Analysis Phase The domain analysis phase has been the most time consuming activity, since the documents of the vendors use different terminology. Guessing common and variant features required a large amount of human inspection. The standards gave support in giving a common language for interpreting the documents and also for the definition of the global feature diagram. Nevertheless, we argue that the domain analysis would benefit from the usage of automated approaches for the identification of common and variant features. We are currently experimenting with a natural language processing approach based on contrastive analysis for the identification of domain-specific terms and the identification of commonality and variability candidates. The current results with the approach, presented in [27], are rather promising. With the automated method we have been able to find 19 commonality candidates, and 6 out of 19 have been considered as common features. Furthermore, we have found 372 variability candidates, and 174 out of 372 have been considered variant features. We argue that the approach would have greatly helped in guiding the inspection of the publicly available documents of the vendors.

Expressiveness of the Feature Diagram The global feature diagram has been found to be a powerful tool also to guide the understanding of the brochures of new vendors. Indeed, the CBTC provided by GE Transportation was not evaluated in the initial domain analysis phase, since brochures for such a product were not available yet. Therefore, we have discovered that the feature diagram is not solely a mean to produce new products, but provides also a reference framework to understand products coming from new competitors, as well as a common language to interpret such products.

Semi-Formal vs *Informal* An aspect that has been highly appreciated by our industrial partner is the choice of the modelling languages. The feature model by itself provides an abstract view of the product family that is easily understood by the stakeholders [13]. On the other hand, the block diagram notation and the sequence diagrams defined allow focusing on the essential concepts, even employing a limited number of operators. The project participants had previous industrial experiences with SysML and Simulink/Stateflow [23, 24]. Nevertheless, they have observed that such languages were too complex to be useful in this analysis phase.

Concerning the definition of the system requirements, the usage of the C++ prototype resulted more effective than the semi-formal Rhapsody in enabling the communication with the industrial partner. Indeed, we argue that – during the requirements elicitation phase – it is relevant to have a prototype that is easy to use, and rather close to the expected system. A semi-formal model is probably a better choice when the requirements have been clearly defined, and when the final target is code generation rather than requirements elicitation.

Number of Requirements We have seen that the number of requirements produced with the presented approach is more than three times larger than the number of requirements produced without employing the approach (182 *vs* 51 requirements). Therefore, we can argue that the scenario-based strategy greatly helps in eliciting requirements. Since we did not implement the system yet, we cannot actually demonstrate that the completeness of the ATS requirements is higher w.r.t. the completeness of the ATO requirements. However, we can reasonably say that a higher number of requirements – expressed with the same level of detail – will cover a larger number of functions in the system-to-be.

Constrained Natural Language The requirements of the ATO were not written in a CNL. Nevertheless, when we analysed them with QuARS, we saw that the number of defects was lower, if compared with the defects found in the ATS requirements (written in CNL). Therefore, we can argue that the presented CNL does not reduce the number of ambiguous expressions. Instead, further appropriate analysis – such as the one performed with QuARS – is required. Nevertheless, the requirements produced with CNL appeared much clearer and precise, compared to the ones produced without constraints. Therefore, the CNL will be employed also in the subsequent phases of the project.

The adoption of the CNL for the definition of the system requirements was not straightforward. Though the proposed language is quite simple, it is still a constrained language, and it was initially perceived as a useless hamper to the creativity required during requirements elicitation. However, after one week of practicing, the team acquired confidence with the language, and we have been able to experience its benefits. For example, the team was more keen to write definitions before writing the requirements. Since the language is constrained and inherently produces short sentences, definitions are indirectly encouraged: once a definition is given for a term, one can use the term easily within the CNL. The usage of definitions further reduces the ambiguity of the requirements.

Requirements Quality When we first defined our approach, we did not focus on the production of *verifiable* and *maintainable* requirements. Nevertheless, we noticed that these two quality attributes indirectly occurred in the produced requirements. Indeed, QuARS helps identifying and reducing the number of *vague* terms. We have seen that vague terms are the main source of defects in our specifications. We argue that the absence

of vague terms enables the production of requirements that can be functionally and - most of all - *quantitatively* verifiable. Furthermore, maintainability of the requirements is eased by the scenario-based approach followed. Requirements are maintainable when the corresponding document is well-structured [52]. The structure of the requirements document and the order of the requirements is guided by the scenarios: requirements are normally in the same section of the document when they are derived from the same scenario. We have seen that, when modifications to the requirements are needed, they normally correspond to modifications to the existing scenarios or to new scenarios to handle. Therefore, it is easy to identify those requirements that have to be changed, or the part of the requirements document where it is preferable to place the new requirements. To further improve the structural quality of the specification, we plan to apply approaches based on sequential clustering that are currently under development [26].

8 Related Works

There is a large literature concerning the development methods of train control systems, including CBTC. Below some works are listed that represent the most relevant examples related to our work.

The MODCONTROL [9] project aimed to define a set of generic requirements for a new generation of Train Control and Monitoring Systems (TCMS). There are two common aspects with our work. The first is the usage of different knowledge sources to define the requirements. Indeed, in MODCONTROL, the requirements have been collected from specifications of existing systems, standards or draft specifications from other EU projects. The second is that the requirements are submitted to an analysis process through the QuARS tool in order to detect potential natural language ambiguities. Nevertheless, modelling and product family engineering, which are core elements of our approach, were not practiced within the MODCONTROL project.

The work performed by LS Industrial Systems [53] concerns the software development of a CBTC system by means of a process based on model-driven development principles. In particular, the UML language is used to model the CBTC software, and source code for the model is derived through the IBM Rhapsody tool. Unlike our case, where requirements are represented in textual form and derived from the analysis of existing systems and standards, the authors use a UML notation (Use Cases) to represent the customer requirements, and do not give details concerning the domain analysis phase and the issues related to the compliance to CBTC standards, which are addressed by our work.

In Rampelli and Virivinti [40], the authors attempt to identify design problems of a CBTC system with respect to factors related to reliability and security, and propose a solution. Their way to solve these problems is to identify the possible architectural design patterns to be applied to CBTC. The advantage of using this technique is that the architectures thus obtained can be reused in a wide range of systems. However, also in this case, no details are given concerning the compliance of the produced artifacts with respect to the existing product standards.

Wang and Liu [51] present an approach for developing a CBTC system based on a 3-levels hierarchical modelling of the system. The three levels are the functional model, the behavioural model of the train, and the model of all control actions. To illustrate this approach, authors use SCADE applied to a case study of a specific CBTC subsystem. The SCADE suite helps the author to capture complex specifications with a graphical notation. Nevertheless, the presented case-study is quite limited and more focused on low-level aspects compared to our work where the emphasis is on high-level requirements and product/process-standard adherence.

Essamé and Dollé [17] present the application of the B method in the METEOR project led by Siemens Transportation Systems. According to the authors, the use of the B method to realize the vital software system for the automatic control of the train, called METEOR, is cost-effective if considered in relation with the entire development process of the CBTC system, which includes the validation of the specification and the product certification. The B method allows the authors to achieve an unambiguos high level software formal specification with a code that maintains the properties of the specification. This allows the validation team to focus its efforts on the specification rather than on code.

Yuan *et al.* [54] illustrate a modelling approach and verification of the System Requirement Specifications (SYS-RS) of a train control system based on the Specification and Description Language (SDL). The application of this approach has allowed the authors to identify possible ambiguities and incompatible descriptions in the requirements. Both our work and that of Yuan *et al.* include formalisms to express requirements of a train control system. Nevertheless, while we use a constrained natural language, Yuan *et al.* use the SDL format to express the requirements. SDL is used by the authors because it is intuitive thanks to its simple conceptual basis (communicating extended finite state machines) and to its graphical representation; there are also tools that allow complete code generation directly from SDL descriptions. In our experience, formal models such as those presented in this paper, can be defined only when the modeller has acquired a proper confidence with the type of system that shall be modelled. We argue that a proper confidence with the system can be reached through prototyping and natural language requirements. Therefore, our approach can be regarded as an intermediate step to achieve the goal of associating a formal model to the SYS-RS requirements.

L. Jansen *et al.* [33] illustrate a modelling approach of the European Train Control System (ETCS) based on Coloured Petri Nets (CPN) and the Design / CPN tools. The proposed approach has been developed within a research project for Deutsche Bahn AG and aims to model the ETCS system according to an hierarchical multi-level decomposition. Specifically, three aspects of ETCS are considered and integrated: components, scenarios and functions. The formal model obtained by Jansen *et al.* will be used to check the completeness of the system specifications and to extract test cases. The computed models are then simulated by the Design/CPN tool and each simulation is performed on a sequence of scenarios with function calls.

Tang *et al.* [48] present a scenario-based approach using UML sequence diagrams and model checking to verify the modeling and specification of the Chinese train control system level 3 (CTCS-3). Sequence Diagrams provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications and Model Checking provides a certain level of confidence on verification of system properties.

F. Bitsch [5] introduces a process model for the development of system requirements specifications for railway systems. The goal of this process model is to achieve a system specification, which is unambiguous and easy to understand for all the parties involved. The modeling language used is the Unified Modeling Language (UML). The authors demonstrate how different techniques of risk analysis can be supported by a system model in UML. The choice of Bitsch on UML derives from its compactness, its intuitive understandability and from the fact that is becoming a standard in system and software modeling.

J.Bohn *et al.* [6] illustrate the overall methodology for developing train system applications based on powerful extensions of the Statemate modeling tool from I-Logix Inc (currently distributed by IBM[9]). The extension includes Live Sequence Charts, Model Checking and Automatic Generation of Test Vectors from the Statemate specification model and Scenarios. The approach illustrated is based on the model checker integrated into Statemate. This is a formal method that allows authors to formally establish the correctness between system requirements and system specifications.

While the previous works are mainly concerned with modelling, the works reported below are focused on the usage of Constrained Natural Languages (CNLs).

C. Denger *et al.* [16] show an approach for reducing the problem of imprecision in natural language requirements specifications for embedded systems with the use of natural language patterns, with a metamodel for describing all possible requirements statements, which allow specifying requirements sentences in a less ambiguous, more complete, and more accurate way. Authors do not give details concerning the use of a tool for requirements analysis.

S. Boyd et al. [8] introduce the concept of replaceability as a way of identifying the lexical redundancy within a sample of requirements, that are written using a CNL. Their approach uses Natural Language Processing (NLP) techniques for optimally constraining the lexicon of a CNL and for increasing readability, expressiveness and unambiguity of CNL.

Agung Fatwanto [22] illustrates a new method for specifying software requirements using a CNL. The presented method comprises two aspects: the articulating medium (using CNL) and the structure for specifying software requirements. The refinement phase is done by analyzing and refining the scenario without the use of tools for requirements analysis. A limitation of this method is that, in a scenario-like structure, only the functional requirements can be represented.

The presented papers can be divided into three main groups. The first three works [9,53,40] mainly concern the usage of semi-formal methods or structured approaches. The last three works [16,8,22,48] concern the usage of CNL techniques for requirements specification. The other works [51,17,54,33,5,6] are focused on formal methods. Our work does not strictly employ formal techniques but uses CNL techniques to define requirements specifications, and semi-formal approaches to define a global CBTC model. Our work can be therefore attached to the first and to the second group. Besides other process-related differences, the current paper mainly differs from all the other works for the emphasis given to the product line aspects of the CBTC development. The main novelty is indeed the domain analysis performed, and the process adopted to define requirements for a novel CBTC system. We argue that this approach enables the development of a modular, competitive, and standards-compliant CBTC system.

Semi-Formal vs *Formal* The choice of using a semi-formal approach instead of strictly using formal methods, as done in most of the articles cited, is due to three main factors: the size of the project, the presence of industrial partners, and the current development phase. These factors have led us to consider that a semi-formal approach is lighter, more scalable, and more understandable by every stakeholder compared to formal methods. Furthermore, in the current development phase, requirements for the CBTC components were not completely clear, and an intermediate step involving prototyping was required to *elicit* the requirements. Nevertheless, we argue that the proposed approach is a structured basis for the introduction of formal approaches. Indeed, formal strategies can be applied in the product line phase, to verify the validity of the selected products by using tools such as SPLOT [36]. Moreover, the CNL patterns defined can be regarded as a starting point for the trans-

[9] `http://www-03.ibm.com/software/products/it/it/ratistat/`

lation of the requirements into a formal logic, such as the parametric SOCL logic [20]. Finally, the current SYS-RS for the ATS component, can be used to define a formal model of the system - for example using the UMC language [49] - to be verified against the requirements. Given these observations, we argue that the presented method can be regarded as a fundamental, structured framework for the introduction of a formal layer to support the future development of CBTC systems. We are currently working along this direction.

9 Conclusion

In this paper, results are presented concerning the definition of a global model for CBTC systems. The model is derived from existing CBTC implementations and from the guidelines of international standards, and is represented in the form of a *feature model*. A methodology has been outlined to derive product requirements from the global model. Furthermore, an approach has been presented to derive system requirements in the CENELEC context for the individual systems that compose the CBTC product. Review of each artifact and validation of each phase is also performed in practice, as required by the CENELEC process. Nevertheless, the presented approach mainly focuses on the system definition part of the CENELEC process, and validation aspects are only partially discussed in this paper. We leave this discussion to future publications. Another relevant aspect is the possibility to adapt the current approach to the development of ERTMS/ETCS systems[10]. Challenges related to this adaptation mainly concern the larger amount of standard documents and implementations associated to these systems. Therefore, we argue that the product-line engineering part of our approach, which highly helps in organizing relevant concepts, can be a proper support to give a reference framework for ERTMS/ETCS systems.

The overall method has been considered higly valuable by our industrial partner, who acted as external supervisors for the presented work. The most promising commercial aspect is the value given to (1) the consideration of the competitor's choices, and (2) to the adherence to the standards (both CBTC and CENELEC ones). Though a migration strategy from a CBTC standard to the other is not fully defined yet, we expect the transition to be simplified by the consideration of all the available standards during the functionality identification phase.

Concerning the compliance to the CENELEC standards for the system requirements definition phase, we are currently working along two directions. The first direction is defining a set of natural language patterns to be employed for the definition of technological, performance, safety and RAM requirements. To this end, we plan to analyse the structure of such type of requirements, by considering example requirements of our industrial partner. Furthermore, the approach shall be tuned to produce *testable* requirements, a quality attribute required by the CENELEC norms. Testable requirements are produced when solely system input and output are involved in the requirements. We argue that a precise definition of the system interfaces and communication protocols is required to achieve this goal, and activities are currently ongoing towards this direction with the support of the developed prototype.

Other activities are also planned to enhance the quality of the NL requirements through automated tools. We are currently working on defining automated strategies to identify conflicting requirements, equivalent requirements, and provide properly structured requirements documents [26]. Furthermore, means to manage requirements evolution (e.g., automated version control) are also foreseen.

Acknowledgements

The authors would like to thank Filippo Salotti and Letizia Bellini from ECM s.p.a. (`http://www.ecmre.com/en/index.xhtml`) for playing the role of domain experts during the research activity presented in this paper. This work was partially supported by the PAR FAS 2007-2013 (TRACE-IT) project.

[10] Please refer to `http://www.uic.org` for a complete list of references concerning ERTMS/ETCS systems

References

1. IEC 62290-1: Railway applications: Urban guided transport management and command/control systems. Part 1: System principles and fundamental concepts. 2007.
2. IEC 62290-2: Railway applications: Urban guided transport management and command/control systems. Part 2: Functional requirements specification. 2011.
3. Ansaldo STS. CBTC Brochure. http://goo.gl/3Kmb0, 2011.
4. D. S. Batory. Feature models, grammars, and propositional formulas. In *Proc. of SPLC*, pages 7–20, 2005.
5. Friedemann Bitsch. Process model for the development of system requirements specifications for railway systems. *Workshop on Software specification of safety relevant transportation control tasks*, 2002.
6. J. Bohn, W. Damm, H. Wittke, J. Klose, and A. Moik. Modeling and validating train system applications using statemate and live sequence charts. In *Proceedings of the Conference on Integrated Design and Process Technology (IDPT2002), Society for Design and Process Science (2002)*, 2002.
7. S. Boyd, D. Zowghi, and A. Farroukh. Measuring the expressiveness of a constrained natural language: an empirical study. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 339–349, 2005.

8. Stephen Boyd, Didar Zowghi, and Vincenzo Gervasi. Optimal-constraint lexicons for requirements specifications. In *Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality*, REFSQ'07, pages 203–217, Berlin, Heidelberg, 2007. Springer-Verlag.
9. Antonio Bucchiarone, Stefania Gnesi, Alessandro Fantechi, and Gianluca Trentanni. An experience in using a tool for evaluating a large set of natural language requirements. In Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, *SAC*, pages 281–286. ACM, 2010.
10. CENELEC. EN 50129, Railway applications - Communications, signalling and processing systems - Safety related electronic systems for signalling, 2003.
11. CENELEC. EN 50128, Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems, 2011.
12. CENELEC. EN 50126, Railway applications - the specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - part 1: Generic RAMS process, 2012.
13. G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel. Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, 2001.
14. Paul C. Clements and Linda Northrop. *Software product lines: practices and patterns*. Addison-Wesley Longman, Inc., Boston, MA, USA, 2001.
15. K. Czarnecki and U.W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley, New York, NY, USA, 2000.
16. Christian Denger, Daniel M. Berry, and Erik Kamsties. Higher quality requirements specifications through natural language patterns. In *In Proc. of the IEEE Int. Conf. on Software Sci. Tech. and Eng*, pages 80–91. IEEE Computer Society, 2003.
17. Didier Essamé and Daniel Dolé. B in Large-Scale Projects: The Canarsie Line CBTC Experience. In *Computer Science*, volume 4355/2006, pages 252–254. 2006.
18. F Fabbrini, M Fusani, S Gnesi, and G Lami. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard*, pages 97–105. IEEE, 2001.
19. A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *Proc. of SPLC*, pages 193–202, 2008.
20. Alessandro Fantechi, Stefania Gnesi, Alessandro Lapadula, Franco Mazzanti, Rosario Pugliese, and Francesco Tiezzi. A logical verification methodology for service-oriented computing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21(3):16, 2012.
21. Alessandro Fantechi, Stefania Gnesi, Gioia Ristori, Michele Carenini, Massimo Vanocchi, and Paolo Moreschini. Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, 4(3):243–263, 1994.
22. A. Fatwanto. Specifying translatable software requirements using constrained natural language. In *Computer Science Education (ICCSE), 2012 7th International Conference on*, pages 1047–1052, 2012.
23. Alessio Ferrari, Alessandro Fantechi, Stefania Gnesi, and Gianluca Magnani. Model-based development and formal methods in the railway industry. *IEEE Software*, 30(3):28–34, 2013.
24. Alessio Ferrari, Alessandro Fantechi, Gianluca Magnani, Daniele Grasso, and Matteo Tempestini. The metrô rio case study. *Sci. Comput. Program.*, 78(7):828–842, 2013.
25. Alessio Ferrari and Stefania Gnesi. Using collective intelligence to detect pragmatic ambiguities. In *Requirements Engineering Conference (RE), 2012 20th IEEE International*, pages 191–200. IEEE, 2012.
26. Alessio Ferrari, Stefania Gnesi, and Gabriele Tolomei. Using clustering to improve the structure of natural language requirements documents. In Joerg Doerr and Andreas L. Opdahl, editors, *Requirements Engineering: Foundation for Software Quality*, volume 7830 of *Lecture Notes in Computer Science*, pages 34–49. Springer Berlin Heidelberg, 2013.
27. Alessio Ferrari, Giorgio Oronzo Spagnolo, and Felice dell'Orletta. Mining commonalities and variabilities from natural language documents. In Tomoji Kishi, Stan Jarzabek, and Stefania Gnesi, editors, *SPLC*, pages 116–120. ACM, 2013.
28. GE Transportation. Tempo CBTC Solution. http://goo.gl/KshrR, 2012.
29. Hassan Gomaa. The impact of rapid prototyping on specifying user requirements. *SIGSOFT Softw. Eng. Notes*, 8(2):17–27, April 1983.
30. Claire Grover, Alexander Holt, Ewan Klein, and Marc Moens. Designing a controlled language for interactive model checking. In *Proceedings of the Third International Workshop on Controlled Language Applications*, pages 29–30, 2000.
31. Institute of Electrical and Electronics Engineers. IEEE Standard for Communications Based Train Control (CBTC) Performance and Functional Requirements. *IEEE Std 1474.1-2004 (Revision of IEEE Std 1474.1-1999)*, 2004.
32. Invensys Rail. SIRIUS Brochure. http://goo.gl/YFUiL, 2009.
33. L. Jansen, M. Meyer Zu Horste, and E. Schnieder. Technical issues in modelling the European Train Control System (ETCS) using Coloured Petri Nets and the Design/CPN tools, 1998.
34. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.
35. Ed Kuun. Open Standards for CBTC and CBTC Radio Based Communications. In *APTA Rail Rail Transit Conference Proceedings*, 2004.
36. Marcilio Mendonca, Moises Branco, and Donald Cowan. Splot: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 761–762. ACM, 2009.
37. Rani Nelken and Nissim Francez. Automatic translation of natural language system specifications into temporal logic. In *Computer Aided Verification*, pages 360–371. Springer, 1996.
38. Robert D. Pascoe and Thomas N. Eichorn. What is Communication-Based Train Control? *IEEE Vehicular Technology Magazine*, 2009.

39. Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
40. S. Rampelli and S. D. Virivinti. Architectural Design Pattern Representation For Communications-Based Train Control System (CBTCS). *International Journal of Engineering Research and Technology (IJERT)*, 2012.
41. F. Roos-Frantz. *Automated Analysis of Software Product Lines with Orthogonal Variability Models: Extending the FaMa Ecosystem.* PhD thesis, University of Seville, 2012.
42. Ken Schwaber. *Agile project management with Scrum.* Microsoft Press, 2004.
43. Rolf Schwitter. English as a formal specification language. In *DEXA Workshops*, pages 228–232. IEEE Computer Society, 2002.
44. Siemens Transportation Systems. Trainguard MT CBTC. http://goo.gl/Xi0h0, 2006. The Moving Block Communications Based Train Control Solution.
45. Signalling Solutions Limited. URBALIS Communication Based Train Control (CBTC) Delivery Performance and Flexibility. http://goo.gl/G3hEe, 2009.
46. Jeffrey S. Stover. CITYFLO 650 System Overview. http://goo.gl/e26SZ, 2006.
47. Alistair Sutcliffe. Scenario-based requirements engineering. In *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, RE '03, pages 320–329, Washington, DC, USA, 2003. IEEE Computer Society.
48. W. Tang, B. Ning, T. Xu, and L. Zhao. Scenario-based modeling and verification for ctcs-3 system requirement specification. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 1, pages V1–400–V1–403, 2010.
49. Maurice H ter Beek, Alessandro Fantechi, Stefania Gnesi, and Franco Mazzanti. A state/event-based model-checking approach for the analysis of abstract system properties. *Science of Computer Programming*, 76(2):119–135, 2011.
50. Thales Transportation. Seltrac Brochure. http://goo.gl/OjhvK, 2009.
51. Haifeng Wang and Shuo Liu. Modeling Communications Based Train Control system: A case study. In *Proc. of ICIMA*, pages 453–456, 2010.
52. William M. Wilson, Linda H. Rosenberg, and Lawrence E. Hyatt. Automated analysis of requirement specifications. In *Proc. of ICSE '97*, pages 161–171, New York, NY, USA, 1997. ACM.
53. C.S. Yang, J.S. Lim, J.K. Um, J.M. Han, Y. Bang, H.H. Kim, Y.H. Yun, C.J. Kim, and G. Cho, Y. Developing CBTC Software Using Model-Driven Development Approach. In *Proc. of WCRR*, 2008.
54. L. Yuan, T. Tang, and K. Li. Modelling and Verification of the System Requirement Specification of Train Control System Using SDL. In *Proc. of ISADS*, pages 81–85, 2011.