UNIVERSITÀ DEGLI STUDI DI FIRENZE
Dipartimento di Ingegneria dell'Informazione

Dottorato di Ricerca in
Ingegneria Informatica, Sistemi e Telecomunicazioni
ING-INF/05 Ingegneria Informatica

Agile processes and formal methods in railway systems

Giorgio Oronzo Spagnolo

Ph.D. Coordinator
Prof. Luigi Chisci

Advisors
Prof. Alessandro Fantechi
Dott.ssa Stefania Gnesi

Ciclo XXVIII - 2012-2015

Contents

Introduction

A business subject who decides to enter an established technological market is required to accurately analyse the products of the different competitors. In the case of cheap mass products (e.g., mobiles, laptops), the new company can actually purchase the products and evaluate their features in order to compare them. In the case of expensive, large-scale, and often customized, products (e.g., security systems, intelligent transport systems), the company has to rely on the existing public documentation about the products, since the cost required to purchase the actual products would be prohibitive. In this work, we consider the case of Communications-Based Train Control (CBTC) systems.

Communications-based Train Control (CBTC) is the most recent technological frontier for signalling and train control in the metro market [1, 2]. 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 [3, 4].

Software systems in the safety domain are becoming increasingly crucial and complex. Safety-critical systems are those where any failure is likely to result in the loss of human life or the damage to the environment. Traditionally, the development of safety-critical systems is approached following a rigorous method such as the V-model[5, 6]. Such a process is characterised by emphasis on design and the production of documentation (typically for use by safety engineers or certifying authorities) in each step of the development process. To develop safety-critical systems, organisations are often required to adopt such processes, but their adoption can make it difficult to manage requirements volatility, introduce new and emerging technologies, and can lead to substantial costs in producing and maintaining documentation. Needless to say, *agile* methods are very attractive to software engineers and project managers working in the safety domain, while posing difficulties and challenges to safety engineers working in this domain.

It has been observed in literature that combination of *agile* and formal methods can bring best features of both the worlds [7] which can lead towards a better software development solution. In [8], authors present an evaluation of *agile* manifesto and *agile* development principles to show how formal and *agile* approaches can be integrated and identify the challenges and issues in doing so. The study [9] focuses on a case study in which an *agile* approach was implemented successfully in a regulated environment. They concluded that the *agile* process as it has been adopted and augmented has worked very well in that regulated environment. In [10], authors suggest that *agile* software development can use light weight formal analysis techniques effectively to bring potential difference in creating systems, with formally verified techniques, on time and within budget. In [11], authors argues that the lightweight and iterative approach taken in *agile* methods can improve the development of safety-critical systems. The authors don't argument that *agile* methods are directly applicable to developing safety-critical systems that require certification. Jonsson et al. made an analysis of *agile* practices in the context of software development for the European railway regulated by EN 50128 stand-

ard [12]. They concluded that *agile* practices support some of the objectives and requirements of EN 50128 but most practices must be tailored to fit in a regulated development environment.

In this context this dissertation present the development a safety critical system, with limited knowledge of the domain, in a context with multiple competitors. The safety critical system shall be developed according to standards (process and product standards). At the same time, there is a limited knowledge of the domain. Hence, *agile* methods fit the need of having an in-depth view of the problem, in limited time, and with limited knowledge.

This dissertation is based on the experience acquired inside the project namely "Train Control Enhancement via Information Technology" (TRACE-IT) funded by Tuscany Region. The project concerns the specification and development of a Communications-based Train Control (CBTC) platform, and sees the participation of the DINFO of the University of Florence, of the Formal Methods and Tools Laboratory of the "Institute of Information Science and Technologies" (ISTI), an institute of the "Italian National Research Council" (CNR) and E.C.M. s.p.a., an industrial partners.

Chapter 1, after an overview concerning CBTC operational principles, presents the overview of the approach and describes the experience to develop a prototype for a CBTC subsystem, starting from the Product families definition and the following *agile* development. Chapter 2 presents how to use natural language processing techniques both to support the feature model definition process and to improve completeness of the requirements with respect to the input documents. Chapter 3 presents how to develop sound solutions based on formal methods to address the problem of deadlock avoidance in our CBTC subsystem prototype.

Chapter 1

Product Families and Agile methodologies

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 [2], while the second is IEC 62290 [13, 14]. 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 systems developed for Europe shall be also compliant with the CENELEC standards [5, 15, 6]. 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 [16], Alstom [17], Thales [18], Invensys Rail Group [19], Ansaldo STS [20], Siemens [21], and GE Transportation [22]. Each vendor provides its own solution, and different technologies and architectures are employed.

In this chapter 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* [23, 24, 25], 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 [26], aided with rapid prototyping [27]. A constrained natural language and natural language processing techniques [28] 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 chapter to explain the approach, and to show the results of the current implementation of the proposed methodology.

In Sect. 1.1, the CBTC operational principles are presented. In Sect. 1.2, an overview of the approach is given. In Sect. 1.3, an analysis of the standards

and of the architectures of the CBTC vendors is presented. In Sect. 1.4, the global CBTC model is described. In Sect. 1.5, the architecture and scenario models are derived, together with the requirements for the actual product. In Sect. 1.6, the approach for the definition of the requirements for the individual systems that compose the CBTC product is presented. Sect. 1.7 describes the current experience with the implementation of the method.

1.1 Communications-based Train Control Systems

CBTC systems [1, 2] 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 1.1).

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 [29].

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* prin-

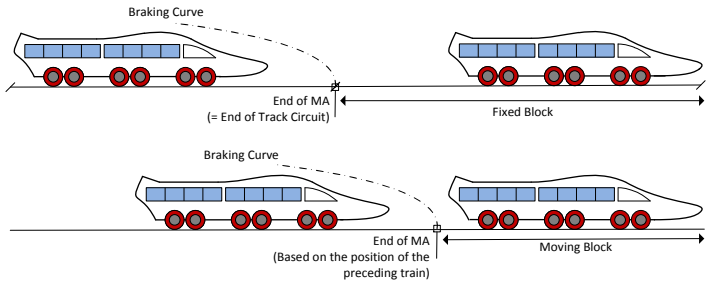


Figure 1.1: Fixed block *vs* moving block

ciple: 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 1.1).

The control system is aware at any time of 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.

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.

1.2 Method Overview

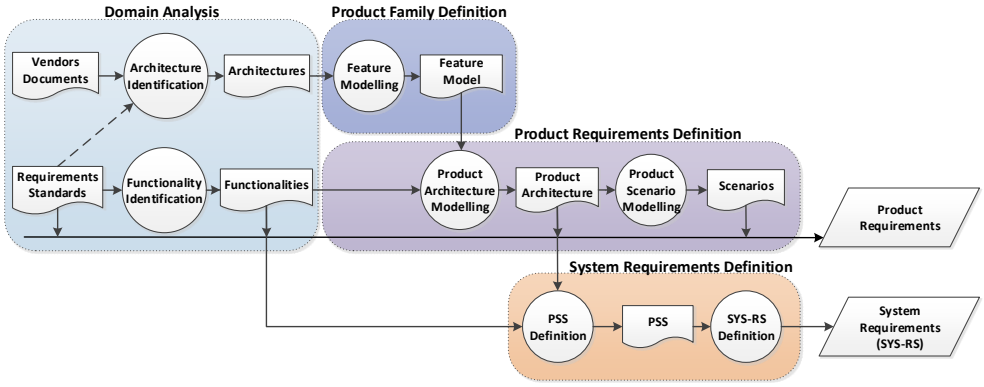


Figure 1.2: Overview of the product requirements definition process adopted

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 [2] and IEC 62290 [13, 14] – 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 1.2 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. 1.3). During this phase, the requirements standards are analysed together with the documents of the dif-

ferent 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. 1.4). 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. 1.5). 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. 1.6). This phase is oriented to accomplish the process-level requirements prescribed by the CENELEC norms [5, 15, 6]. First, a Preliminary System Specification (PSS) document is defined, which is based on the functionalities extracted from the product standards and on the chosen product architecture (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 required (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 [30], while the QuARS tool [28] 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 chapter.

1.3 Domain Analysis

The Domain Analysis phase is composed of two sub-phases, 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.

1.3.1 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 [2] and IEC 62290 [13, 14], which are briefly summarized below.

[1] `http://office.microsoft.com/en-us/word/`
[2] `http://office.microsoft.com/en-us/excel/`
[3] `http://office.microsoft.com/en-us/visio/`
[4] `http://www.microsoft.com/visualstudio/eng/visual-studio-2013`

1.3.1.1 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).

1.3.1.2 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 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.

1.3.1.3 Functionalies

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 1.7 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 chapter, 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;

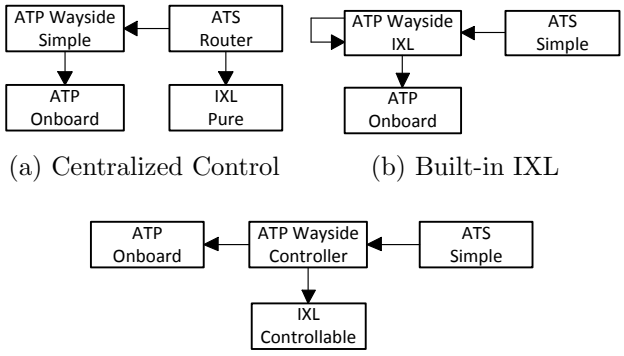
Train Identification and Tracking. (ATS - IEEE 6.3.3) This functionality monitors the position and the identity of the trains.

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

1.3.2 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 [16], Alstom [17], Thales [18], Invensys Rail Group [19], Ansaldo STS [20], Siemens [21], and GE Transportation [22].



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

(c) Controllable IXL

Figure 1.3: Architectures extracted

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 1.3a) 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 1.3b) 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 1.3c) 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 1.3a and 1.3b are both used by Bombardier. The second architecture is described in the Bombardier documentation as CITYFLO 650 with built-in IXL. Though architecture 1.3a is not explicitely described, the Bombardier documentation states that, when available, the IXL works as a

backup system in case of ATP failure. Therefore, we can argue that the IXL control resides in the ATS and not in the ATP.

Architecture 1.3c 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 1.3c. 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.

1.4 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 [4, 3]. This development cycle aims at lowering the development costs by sharing an overall reference architecture for all products. 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 [31]. 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 [31].

1.4.1 Feature Modelling

The modelling of variability has been extensively studied in the literature, with particular focus on *feature modelling* [23, 24, 25]. 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 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 [24, 32].

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 [23]. It is a tree where primitive features are leaves and compound features are internal nodes. Common graphical notations are depicted in Figure 1.4.

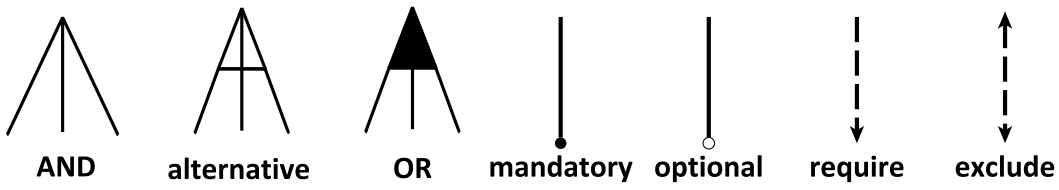


Figure 1.4: Feature diagram notations

1.4.2 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

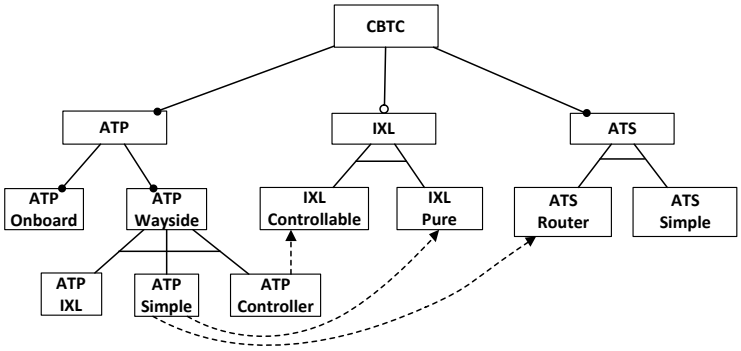


Figure 1.5: Simplified excerpt of the CBTC global feature diagram

(Sect. 1.3.2). We show the model for the GoA-1 level, according to the IEC 62290 terminology [13]. In other terms, such a model assumes the presence of a driver on board (i.e., there is no ATO system).

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 1.5. The diagram includes the architectural components (which in our diagram become *features*) already identified in Sect. 1.3.2.

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` sub-tree 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 [33], 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.

1.5 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 1.3c. 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 family engineering paradigm. Architecture and scenarios are employed to derive requirements for the actual

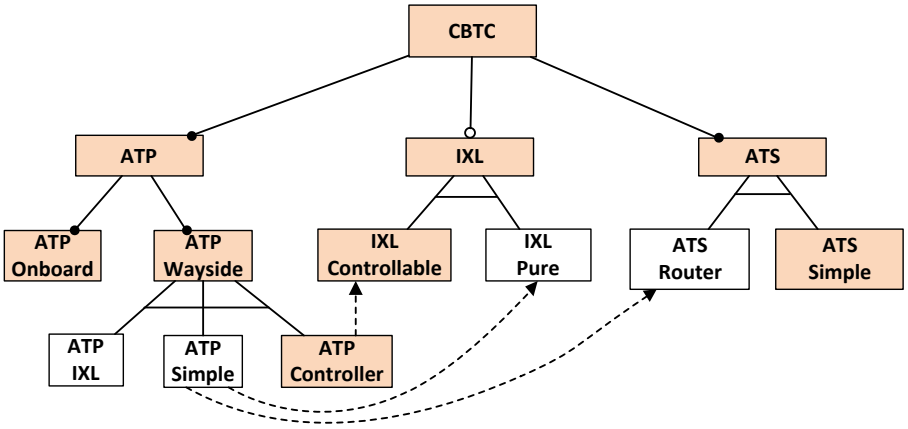


Figure 1.6: Selection of features for our example product

product.

1.5.1 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 [34, 35]. 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 1.6 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 1.3c).

An excerpt of the detailed architecture for the selected product is depicted in Figure 1.7. 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.

1.5.2 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 architec-

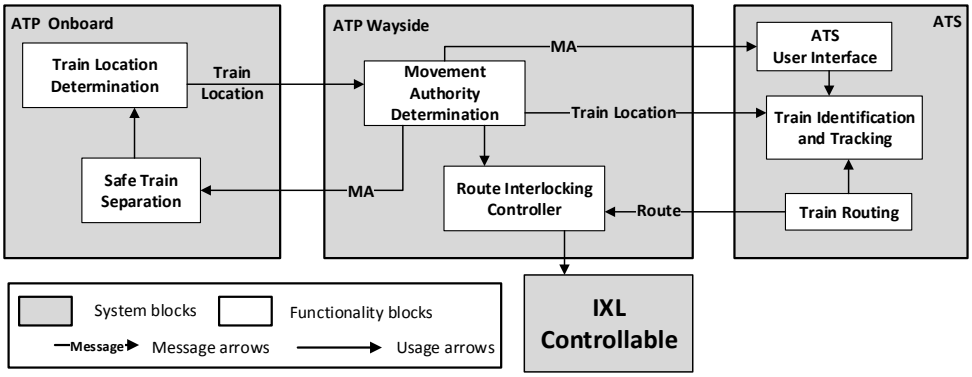


Figure 1.7: Architecture example for a CBTC system

tural 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 *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 1.8 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.

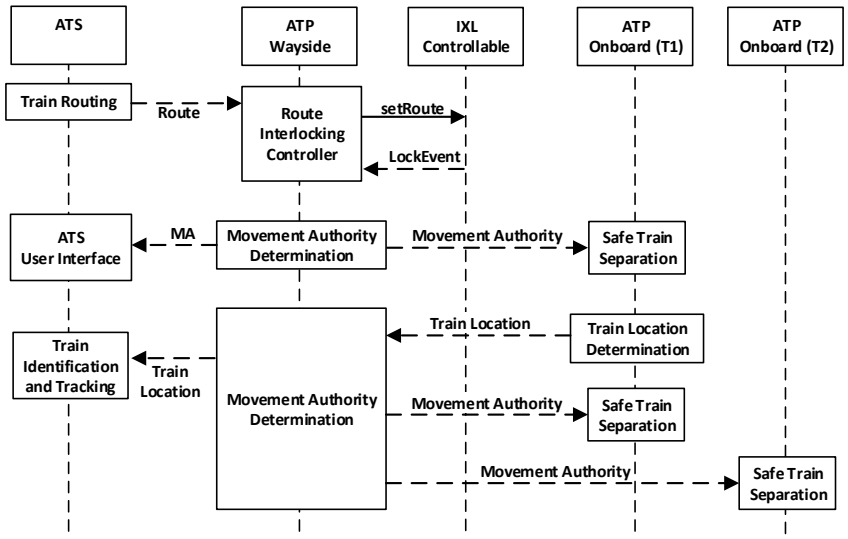


Figure 1.8: Example sequence diagram: a train moves from one station to another

1.5.3 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 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.

1.6 System Requirements Definition

The development of railway and metro signalling platforms in Europe shall comply with the CENELEC standards [6, 5, 15]. 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 sub-system 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 [26], aided with rapid prototyping [27] to produce the SYS-RS document. Moreover, our method expects the SYS-SRS document to be produced in a constrained natural language.

1.6.1 PSS Definition

The approach for the definition of the PSS is as follows. First, we select from the product architecture the sub-system to be developed. We choose, for example, the `ATS Simple` introduced in Sect. 1.3.2 and employed in the example of Sect. 1.5. The information provided by the detailed architecture diagram for the ATS is the same information required by the PSS document. The *message 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.

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 1.1: Excerpt of the interfaces of the ATS sub-system

In Table 1.1, we give an excerpt of the PSS of the ATS sub-system concerning the interfaces with the other sub-systems 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.

While the functionalities are extracted from the detailed architecture diagram, the natural language details concerning such functionalities can be dir-

ectly extracted from the Domain Analysis Phase (see Sect. 1.3.1). 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. [*continue...*]

- 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 [*continue...*]
- F4.3. Provide interface to the operation control centre HMI (IEC 6.2.2.5.1)) [*continue...*]
- F4.4. Provide interface to the decentralized HMI (IEC 6.2.2.5.2) [*continue...*]

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.

1.6.2 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* [5]. 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 [28], in order to produce unequivocal requirements.

Figure 1.9 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 pos-

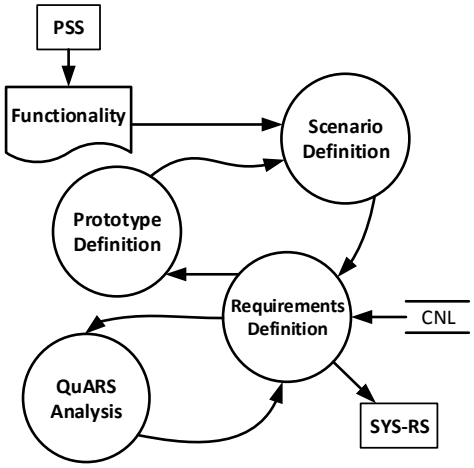


Figure 1.9: Approach for the definition of the SYS-RS

sible 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 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. 1.5.2, 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 1.2, we report one of the scenarios that have been derived from the functionality F4.1. *Management of*

the Train Describer.

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 1.2: Example scenario 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 [36, 37] for some examples, and [38] 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 [39]. 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 require-

ments 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 position of 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 [28]. 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 [40].

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 [41]:

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

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

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

parametric formulas is UMC [42], which is also available on-line[6]. Other experiences have been presented in the literature that aim at transforming natural language requirements into formal specifications (e.g., [43, 44]). 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.

1.6.2.1 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.

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

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

sends the message to the ATS prototype;

- a communication interface that receives the ATP messages;
- a graphical user interface that represents the Train Describer.

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 distance, the latter starts at L distance, and $L - M > \tau$.

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

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 account 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.

1.6.3 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 1.10 depicts the traceability links enforced by our approach. The Functionalities extracted from the 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

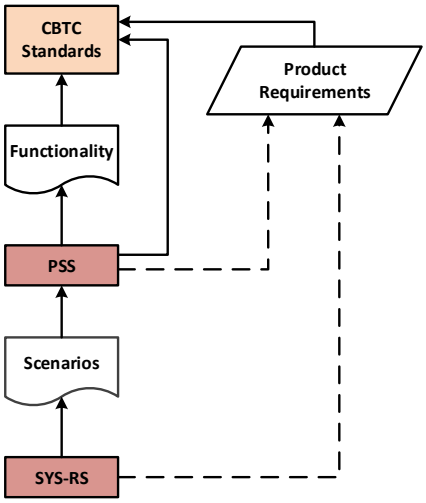


Figure 1.10: Traceability links among artifacts in the system definition phase (solid line=explicit link, dashed line= manual link)

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.

1.7 Experience Report

The approach presented in this chapter 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 chapter 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 chapter, 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 1.3 summarizes the number of functionalities associated to each sub-system.

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

Table 1.3: Number of functionalities of the standards associated to each subsystem (ATP W. = ATP Wayside, ATP O. = ATP Onboard).

Then, the documents of the vendors have been evaluated and a global feature model was derived, as described in Sect. 1.4. A product instance has been chosen from the 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 1.11.

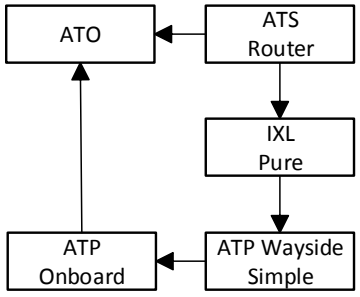


Figure 1.11: Architecture of the chosen product instance

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. 1.5. For confidentiality reasons, the examples reported in this section of the chapter 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 chapter, 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. 1.6, 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 [45]. 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

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

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

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".

1.7.1 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 [30], 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 [46]. 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 [35, 34]. 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 [47]. 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 [48].

Chapter 2

Natural Language Processing approaches

Natural Language Processing (NLP) began in the 1940s as the intersection of artificial intelligence and linguistics analysis. NLP is the computerized approach to *analyzing sentences in a natural language* that is based on both a set of theories and technologies [49]. The advances in natural language processing provide ample opportunities for the documents in a natural language to be analyzed and mined, thus creating numerous new and valuable applications. This chapter presents a set of Natural Language Processing approaches to extract information from the documents, both to semi-automate the process to define a *product family* (Section 2.1) and to measure and improve the *backward functional completeness* (Section 2.2) of natural language requirements documents. These approaches are used to improve the product requirements definition process adopted in the previous Section 1.2.

2.1 NLP approach to Product Family Definition

In the previous Chapter a set of publicly available documents (*brochures*) has been used to derive a global model, from which specific product requirements for novel systems belonging to the same product line have been derived. The goal of the model was to support the analysis of available Communications-based Train Control Systems (CBTC) products, which are integrated platforms to control the movement of trains within a station and across different stations (see Section 1.1). The model was represented in the form of a *feature diagram* [23], following the principles of the product line engineering technology. The bottleneck found in the experience was the large amount of human inspection required to identify the common components, as well as the architectural differences, between the solutions proposed by the different vendors. The identification of these *commonalities* and *variabilities* has enabled the definition of mandatory and optional features in the global feature diagram. In order to reduce the time required to extract commonalities and variabilities from the brochures of the different vendors, in [30] we suggested to adopt an automated Natural Language Processing (NLP) approach named *contrastive analysis* to identify *domain-specific terms* (single and multi-word) from textual documents [50]. The proposed method takes the brochures of the different vendors as input, and identifies the linguistic expressions in the documents that can be considered as *terms*. In this context, a *term* is defined as a conceptually independent expression. The domain-specific terms that are common among all the brochures are considered as *commonality candidates*. On the other hand, those domain-specific terms that appear solely in a subset of the brochures are considered as *variability candidates*. Starting from the experiences presented in previous chapter and [30], two graphical tools have implemented that (1) support the extraction of commonalities and variabilities from natural language (NL) documents, and (2) allow to graphically design a feature model based on the extracted commonalities and variabilities. The first goal is addressed by the Commonality Mining Tool (CMT), while the second goal is addressed by the Feature Diagram Editor (FDE). Though the definition of the

two tools is based on an experience focused on brochures, we advocate that the tools can be used whenever the feature model has to be defined starting from any type of NL documents, including NL requirements. Both the tools are freely available at `https://github.com/isti-fmt-nlp/tool-NLPtoFP`.

The section is organised as follows. In Subsect. 2.1.1, we list the main characteristics of the two tools, and their architecture. In Subsect. 2.1.2, we describe the NLP approach based on contrastive analysis to identify commonality and variability candidates. In Subsect. 2.1.3, we describe the details of the two tools.

2.1.1 Overview

Commonality Mining Tool (CMT) allows commonalities and variabilities from NL brochures of existing products to be extracted. The main functionalities of CMT are:

1. **Terminology Extraction:** given a set of documents belonging to different vendors, the tool allows the automatic extraction of the domain-specific terms, namely the specific words related to the domain of the product, from each document;

2. **Commonality Candidates Extraction:** the tool automatically identifies of the commonality candidates among the domain-specific terms. These are the domain-specific terms appearing in *all* the documents;

3. **Variability Candidates Extraction:** the tool automatically identifies the variability candidates among the domain-specific terms. These are the domain-specific terms that appear only in a sub-set of the documents;

4. **Documents Surfing:** the user can verify the quality of the selected candidates, by searching the occurrences of candidates in the original documents through the Graphical User Interface (GUI) of CMT;

5. **Commonality/Variability Selection:** among the candidates, the user can select the commonalities and variabilities for the construction of a feature model, manually adding others if needed.

Feature Diagram Editor (FDE) is a tool to define a feature model through the construction of its graphic representation, namely the feature diagram. The main functionalities of FDE are:

1. **Feature Diagram Generation:** the tool automatically defines an initial feature diagram with a set of features selected by the user, based on the commonalities and variabilities produced by CMT;

2. **Feature Diagram Editing:** the user can create, edit and save a feature diagram through a graphical interface based on Drag&Drop operations.

3. **Feature Diagram to Documents Surfing:** the user is guided in surfing the input documents – the same used by CMT – to search for occurrences of features;

4. **SPLOT Import:** the user can import the description of a feature model from the XML format generated by the online tool SPLOT[1] [51] (*.sxfm format). The feature model is automatically rendered in a feature diagram;

5. **SPLOT Export:** the user can export the feature model in the SPLOT format and in *.png image format.

The architecture of the two tools and their interaction is shown in Fig. 2.1. The user interacts with CMT through an intuitive GUI (**CMT GUI**). From the GUI, the user can load the natural language **Brochures** of different vendors and can perform terminology extraction, commonality/variability candidates extraction, document surfing and commonality/variability selection.

The internal engine of CMT (**Commonality/Variability Analyser**) interacts with an external tool named **T2K** [52]. The tool is in charge of performing

[1] `http://www.splot-research.org`

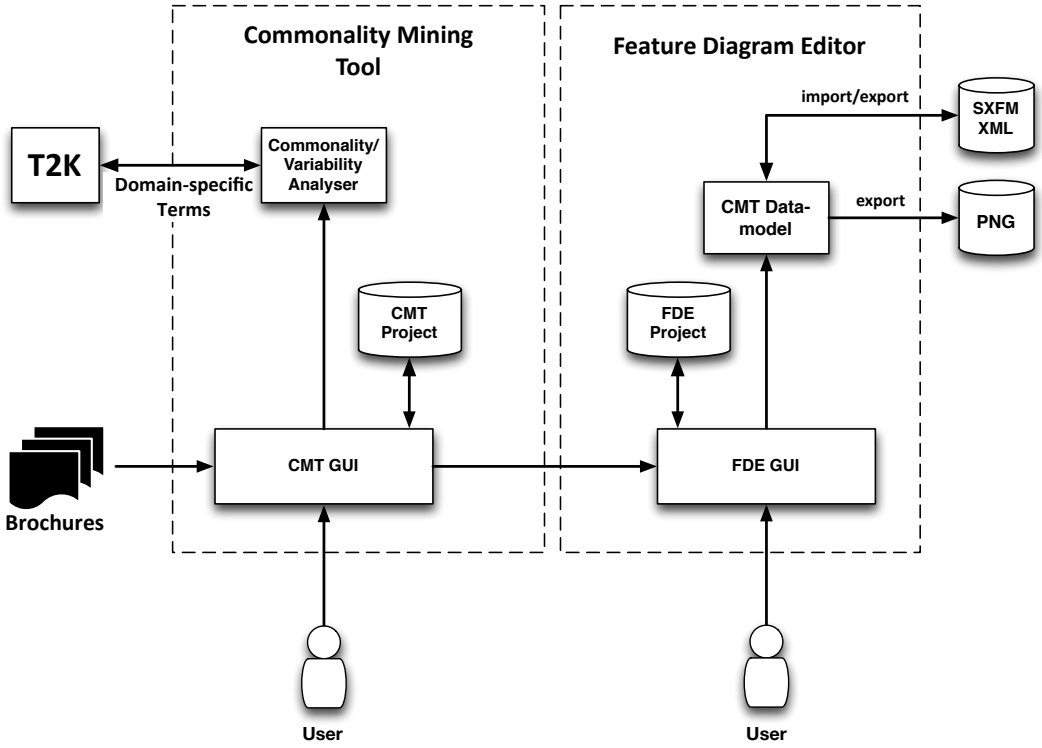


Figure 2.1: Architecture and interactions of CMT and FDE

the terminology extraction, and other NL analysis of the text included in the brochures. CMT allows to store the analysis in a **CMT Project**, which can be saved and loaded by the user.

From CMT, the user can launch FDE. In this case, FDE takes as input the commonalities and variabilities extracted by CMT and stored in the CMT Project. Moreover, a textual version of the original documents is also passed to FDE. The user can interact with the GUI of the tool (**FDE GUI**) to edit the diagram, surf the documents from the features represented in the diagram, or import/export the feature model in the SPLOT format (**SXFM XML**). Moreover, the user can save and load a feature diagram in a **FDE Project**, which includes an XML version of the diagram. FDE can also be executed by the user as a standalone application. In this case, an empty FDE Project is created and the user can start editing the diagram from scratch.

2.1.2 The NLP Approach

The method employed by CMT, and supported by T2K [52], is based on a novel natural language processing approach, named *contrastive analysis* [50], for the extraction of *domain-specific terms* from natural language documents. In this context, a *term* is a conceptually independent linguistic unit, which can be composed by a single word or by multiple words. For example, "Automatic Train Protection" is a term, while "Protection" is not a term, since in the textual documents considered in the study reported in [30] it often appears coupled with the same words (i.e., "train", "mission"), and therefore it cannot be considered as conceptually independent.

The *contrastive analysis* technology aims at detecting those terms in a document that are *specific* for the domain of the document under consideration [50, 53]. Roughly, contrastive analysis considers the terms extracted from domain-generic documents (e.g., newspapers), and the terms extracted from the domain-specific document to be analysed. If a term in the domain-specific document highly occurs also in the domain-generic documents, such a term is considered as domain-generic. On the other hand, if the term is not frequent

in the domain-generic documents, the term is considered as domain-specific.

In our work, the documents from which we want to extract domain-specific terms are the brochures of different vendors. A brochure is a promotional document that describes the product to possible customers. Here, the reasonable assumption is that both commonalities and variabilities can be found among the domain-specific terms of the brochures. The proposed method is summarized in Fig.1.2. First, conceptually independent expressions (i.e., *terms*) are identified (**Identification of Terms**). Then, **Contrastive Analysis** is applied to select the terms that are domain-specific. From these terms, commonality and variability candidates are extracted (**Commonality/Variability Candidates Identification**). In the tools presented in section 2.1.3, the former task is supported by T2K, while the second task is in supported by the Commonality/Variability Analyser component of CMT.

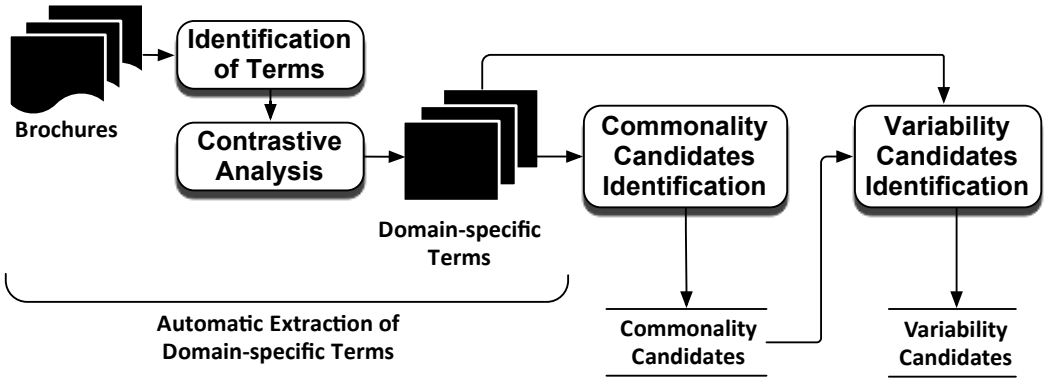


Figure 2.2: Overview of the approach

2.1.2.1 Identification of Terms

Each vendor might have more than one brochure. We collect the brochures of the same vendor i in a single document D_i. Therefore, given n vendors, we have $D_1 \dots D_n$ documents. From each one of these documents we identify a ranked list of *terms*. To this end, we perform the following steps.

POS Tagging: first, Part of Speech (POS) Tagging is performed with an english version of the tool described in [53]. With POS Tagging, each word is

associated with its grammatical category (*noun*, *verb*, *adjective*, etc.).

Linguistic Filters: after POS tagging, we select all those words or groups of words (referred in the following as *multi-words*) that follow a set of specific POS patterns (i.e., sequences of POS), that we consider relevant in our context. For example, we will not be interested in those multi-words that end with a preposition, while we are interested in multi-words with a format like <*adjective*, *noun*, *noun*> (such as "Automatic Train Protection").

C-NC Value: terms are finally identified and ranked by computing a "termhood" metric, called C-NC value [50]. This metric establishes how much a word or a multi-word is likely to be conceptually independent from the context in which it appears. The computation of the metric is rather complex, and the explanation of such computation is beyond the scope of this section. The interested reader can refer to [50] for further details. Here we give an idea of the spirit of the metric. Roughly, a word/multi-word is *conceptually dependent* if it often occurs with the same words (i.e., it is *nested*). Instead a word/multi-word is *conceptually independent* if it occurs in different context (i.e., it is normally accompanied with different words). Hence, a higher C-NC rank is assigned to those words/multi-word that are conceptually independent, while lower values are assigned to words/multi-words that require additional words to be meaningful in the context in which they are uttered.

After this analysis, for each D_i, we have a ranked list of words/multi-words that can be considered *terms*, together with their ranking according to the C-NC metric, and their frequency (i.e., number of occurrences) in D_i. The more a word/multi-word is likely to be a *term*, the higher the ranking. From the list we select the k terms that received the higher ranking. The value of k shall be empirically selected. A higher value guarantees that more domain-specific terms are included in the list. On the other hand, higher values for k might also introduce noisy items, since also words/multi-words with low rank might be included.

2.1.2.2 Contrastive Analysis

The previous step leads to a ranked list of k terms where all the terms might be *domain-generic* or *domain-specific*. With the contrastive analysis step, terms are re-ranked according to their domain-specificity. To this end, the proposed approach takes as input: 1) the ranked list of terms extracted from the document D_i; 2) a second list of terms extracted with the same method described in Sect. 2.1.2.1 from a set of documents that we will name the *contrastive corpora*. The contrastive corpora is a set of documents containing domain-generic terminology. In particular, we have considered the Penn Treebank corpus, which collects articles from the Wall Street Journal. The reasonable assumption here is that a term that frequently occurs in the Wall Street Journal is not likely to be a domain-specific term of the metro domain. The new rank $R_i(t)$ for a term t extracted from a document D_i is computed according to the function [50]:

$$R_i(t) = (\log(f_i(t)) \cdot (\frac{f_i(t) \cdot N_c}{F_c(t)})$$

where $f_i(t)$ is the frequency of the term t extracted from D_i, $F_c(t)$ is the sum of the frequencies of t in the contrastive corpora, and N_c is the sum of the frequencies of all the terms extracted from D_i in the contrastive corpora. Roughly, if a term is less frequent in the contrastive corpora, it is considered as a *domain-specific term*, and it is ranked higher. If two terms are equally frequent in the contrastive corpora, but one of them is more frequent in D_i, it is considered as a term that characterizes the domain more than the other, and, again, it is ranked higher.

After this analysis, for each D_i, we have a list of terms, together with their ranking according to the function R, and their frequency in D_i. The more a term is likely to be domain-specific, the higher the ranking. From each list, we select the l terms that received the higher ranking. The choice of l shall be performed empirically: higher values of l tend to include terms that are not domain-specific, while lower values tend to exclude terms that might be relevant in the subsequent phases.

2.1.2.3 Commonality Candidates Identification

The commonality candidates are the domain-specific terms that are common to all the documents. Indeed, if a term is *domain-specific* and appears in all the documents of the different vendors, it is likely to be a common feature of all the products. More formally, if $C_1 \dots C_n$ are the sets of domain-specific terms for $D_1 \dots D_n$ respectively, then the set of commonality candidates is defined as: $C = \{C_1 \cap C_2 .... \cap C_n\}$. Ranking is provided also for the set of commonality candidates. The ranking value is provided by computing the average rank of each term.

2.1.2.4 Variability Candidates Identification

The variability candidates are identified as those terms which are domain-specific, and therefore appear in some of the C_i sets, but are not part of the commonalities. We assume that, if a domain-specific term appears in some of the documents of the different vendors, but not in all of them, it is likely to be a variant feature, characterizing only a sub-set of the products. More formally, we define the variability candidates as $V = \{C_1 \cup C_2 \dots \cup C_n\} \setminus C$. Also in this case, the ranking value is provided by computing the average rank of each term.

The sets C and V are domain-specific terms of the documents. In order to assess that they actually include commonalities or variabilities, a human operator shall assess the actual relevance of each candidate.

2.1.3 CMT and FDE

In this section we describe the functionalities of the Commonality Mining Tool (CMT) and of the Feature Diagram Editor (FDE). The former employs the approach explained in the previous section to extract commonality and variability candidates, with the support of the tool T2K for domain-specific term extraction (also referred as "terminology extraction" in the following). The latter is used to build a feature diagram.

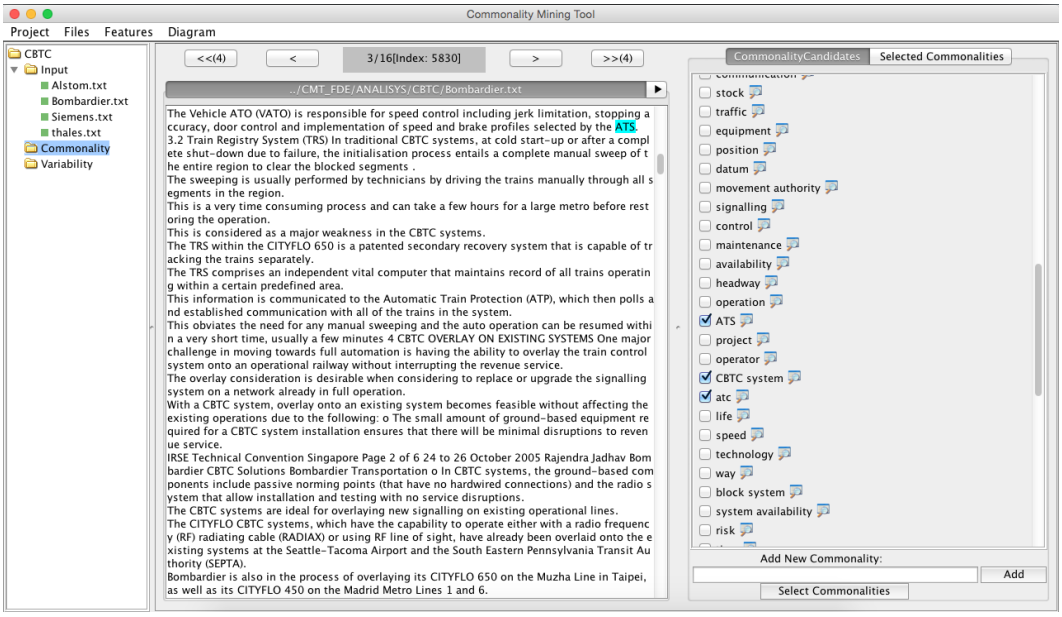


Figure 2.3: Commonality Mining Tool - The user can surf the original documents to check occurrences of the features in the text. The checked candidates (right panel) are the features that will be passed to FDE.

2.1.3.1 Commonality Mining Tool

The Commonality Mining Tool (CMT) provides the extraction of feature candidates starting from the information contained in NL documents that describe similar products. Moreover, among the feature candidates, the tool extracts common and variant feature candidates, to be later evaluated by a human operator (referred in the following as the user).

The idea is to start from a set of NL documents, in pdf/txt format, and extract the set of domain specific terms from these documents. To this end, CMT relies on **T2K (Text-To-Knowledge)** tool [52], which is specifically targeted to identify domain-specific terms.

Once fed with NL documents as input, T2K will provide a set of files containing:

- the NL documents, in txt format;
- the separation into sentences;
- the terminology extraction (i.e., the list of domain-specific terms ranked by relevance);
- the annotation of the text according to the grammar analysis (POS Tagging).

These files will be used to extract the commonality candidates (i.e. domain-specific terms that appear in each document) and variability candidates (all other domain-specific terms). Moreover, the separation into sentences, and the documents in *.txt format will be used to support the identification of relations among the different domain-specific terms extracted.

2.1.3.2 How CMT Works

A screen-shot of the visual interface provided by CMT is shown in Fig. 2.3. The internal process followed by CMT can be summarised in the following phases.

Project Set-up In this phase, the user creates a CMT Project and loads the NL brochures in *.txt/*.pdf format. The tool assumes that for each vendor, a single document is loaded. Therefore, the user is in charge of merging the different NL documents (through copy/paste or supported by external tools) into a single document. The tool will create a folder for each vendor, which will be used to store the different analysis performed later on.

Terminology Extraction In this phase all the NL documents are given as input to CMT, each of them associated to a different folder. For each folder the tool reads the domain-specific terms as they have been processed by T2K. Then, it identifies and stores the position of these terms in the source document(s), and stores the separation into sentences, to be used in the *Color by Cluster* phase of the process described in the following paragraphs.

Extraction of Candidates This phase provides the extraction of commonality and variability candidates as follows. Let $D_1 \dots D_n$ is the set of NL documents and T_i the set of relevant terms extracted from the document D_i.

Commonality candidates are computed as:

$$Commonality\ Candidates = \bigcap_{i=1}^{n} Ti$$

Variability candidates are computed as:

$$Variability\ Candidates = \bigcup_{i=1}^{n} Ti - \bigcap_{i=1}^{n} Ti$$

Color by Cluster In this phase, colors are assigned to the feature candidates (commonalities and variabilities) to ease the job of the user in understanding the relations among the different features, when such features will be visually shown in FDE. The idea is to assign the same color to *variabilities* that have a textual relation in the original documents. Features associated to domain-specific terms that occur in neighbouring sentences are considered to have a textual relation. Instead, all *commonalities* will be associated to the same color (black, in the default configuration).

To assign colors that highlight relations among variabilities, the position of all the domain specific terms is identified in all the input documents. Such occurrences are used to group the terms in a fixed number of clusters. A cluster identifies a set of terms that have a relation. Here, we use the generic term "relation", without specifying the type of relation, since the relations that we highlight are based solely on the distance of terms within the text. The user will be then in charge of establishing the actual type of relation that occurs among the colored terms: such relation can be a hierarchical one (parent/child feature), a AND/OR relation, or a constraint such as exclude or require. Moreover, such relation can also not exist, since the color highlights relations based on distance in the text, which could not match with semantics relations in the final feature model.

The clustering algorithm adopted to assign colors to clusters is loosely based on K-Nearest Neighbours [54]. A color identifier is assigned to each cluster, which will be associated to all of its terms. The colors will be used by FDE to visualize features that belong to the same cluster. Without going into the details of the algorithm, the reader should imagine that, if two terms are frequently occurring in sentences that are close one to the other, then the terms will be associated to the same color.

The colors associated to each term can be visualised by the user through CMT, but the user will be able to modify the different colors assigned by the algorithm only through FDE. The coloring feature shall be regarded as a *recommendation* of the tool-suite to the user, who will be free to change colors and add new colored features in FDE. Within this work-flow, we do not enforce strict consistency between the colors of the final feature model, and the colors originally generated. Indeed, the goal here is just to *suggest* relations among features in the text, and not to constrain the activity of the user in designing the feature model.

Feature Selection During this phase, the user visualises the commonality and variability candidates, checks their occurrences in the input documents, and selects those that seem to be appropriate for the construction of the feature

diagram. The user can also manually add other features that s/he thinks necessary. In this phase the user can surf the original documents, by searching the occurrence of a candidate within the text. For example, in Fig. 2.3, the user is looking at one occurrence of the commonality candidate named "ATS" in one of the original documents. The checked candidates in the right panel of the figure are those that the user has selected as actual commonalities that will be sent to FDE. When the user presses the "Select Commonalities" button at the bottom-right of Fig. 2.3, the checked candidates becomes visible in the "Selected Commonalities" tab (activated by clicking on the top-right button of Fig. 2.3). Similar panels and approaches are provided for variability candidates.

CMT allows searching only one term at a time, and one occurrence of term at a time, to enable accurate inspection of the documents. The search of term occurrences is designed to "remember" the last searched term. In this way, the user can return to such term if, after other searches, there is the need to consider again that term. This functionality is important for the usability of the tool, in order to help discarding the unnecessary terms, and to enable reasoning on the extracted terms by looking at their textual context.

Diagram Generation Now the user can run FDE to begin the construction of the feature diagram. If launched by CMT, the commonalities and variabilities selected by the user will be passed to FDE, together with their colors – as assigned by CMT – and their positions and occurrences in the input documents. The tool FDE builds an initial diagram with a root with the same name of the project created with CMT. The selected features are shown as children of such root.

2.1.3.3 Feature Diagram Editor

The Feature Diagram Editor aims to define a feature model through the construction of its graphic representation, namely the feature diagram. The *feature diagram* notations are used by Feature Diagram Editor and it's presented

in Section 1.4.1. A user can start interacting with FDE according to three workflow starting points:

- **from CMT**: in this case the selected features will be given in input, together with their colors and the information about their position in the original texts;
- **as a standalone application**: in this case, the user can edit the diagram from scratch without relying on previously extracted features;
- **importing an SXFM files**: an SXFM file is an .xml file generated with the tool SPLOT [51]. In this case, FDE will automatically generate the Feature Diagram corresponding to the feature model defined in such file.

Basic Operations FDE is used mainly by means of Drag&Drop operations. Fig. 2.4 shows the interface of the tool[2] (ignore at this stage the "Search Feature" label in the figure). FDE has a palette on the left with the graphical symbols already reported in section 1.4.1 Fig. 1.4 (AND decomposition can be performed by combining the mandatory/optional connectors). The user can select one of the symbols from the palette and drag it to the central dashboard, to build or update the feature diagram. With this user-friendly approach, new features can be introduced, as well as connections among features.

Some functionalities of FDE are activated by means of a pop-up menu that can be opened by right clicking on a feature. Among them, the change of the name of the feature, or the opening of a window to search occurrences of the feature in the original documents.

Finally, saving, loading, import and export operations can be accessed through the menu bar of the tool (under the "Files" menu). Here, it is worth noting that, when saved, the visual diagram is mapped to a formal model expressed in XML. When exported in the *.sxfm format, such model can also

[2]The colors of the feature diagram in the figure have been adjusted by the user. Indeed, right after importing the features from CMT, all the commonalities are normally colored in black.

be read by the SPLOT tool [51], which allows performing additional analysis on the product family associated to the model.

Surfing the Documents The workflow of FDE highly depends on the user preferences and needs. However, here it is useful to describe how the user can surf the original documents of the different vendors starting from the visual representation of the feature diagram.

As shown in Fig. 2.4, the user can select a group of features, and right click to search them in the original texts. In Fig. 2.4, the user has selected a group of two features, named "CBTC System" and "ATS" (a component of the CBTC system). When the user presses "Search Feature", FDE opens the window shown in Fig. 2.5. From such window the user can see the occurrences of the selected features in the original documents.

It is worth noting that the colors displayed in this window have a different meaning with respect to those generated by CMT, and shown in the feature diagram. Here, the colors serve to understand whether the feature was extracted from the text as a commonality, a variability or was an additional feature not previously extracted from the text, as shown in the legend at the top-left of Fig. 2.5.

2.1.3.4 Tool Download

CMT and FDE have been developed in Java, to ensure their portability. The source code can be freely downloaded from `https://github.com/isti-fmt-nlp/tool-NLPtoFP`, together with some illustrative examples.

After downloading the tools, which are embedded in a single project, the user can import them as a Maven project[3] within the Eclipse[4] platform. Both tools are under LGPL license. FDE can be executed as-is. Instead, terminology extraction through CMT is performed remotely.

In our experience, manual inspection of the brochures and identification of all the common and variable components of the different architectures of the

[3] `https://maven.apache.org`
[4] `http://www.eclipse.org`

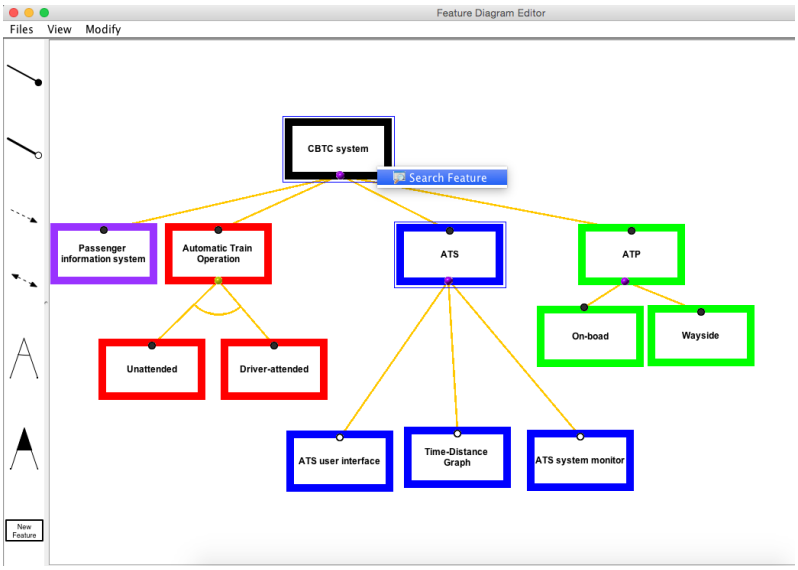


Figure 2.4: Feature Diagram Editor - The tool allows building a feature diagram through Drag & Drop operations, by using the palette on the left and dragging the graphical elements to the central dashboard.

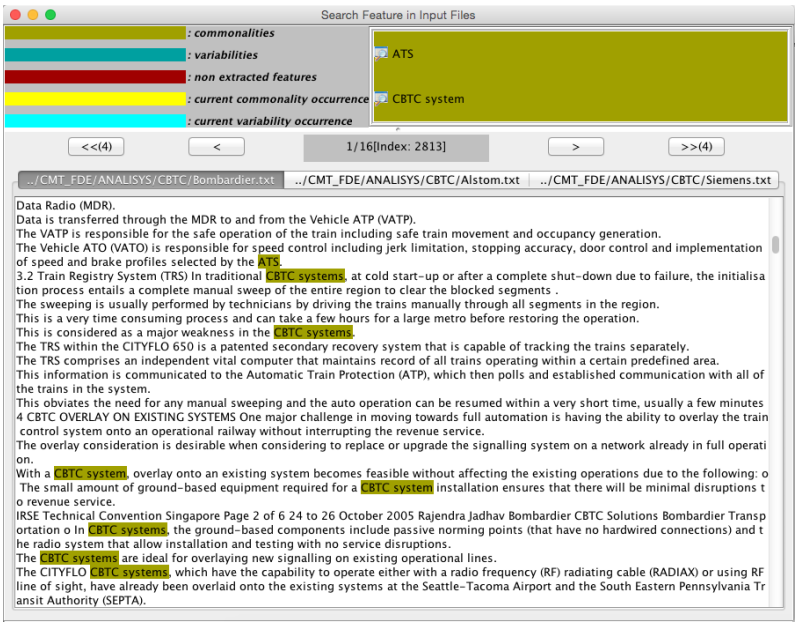


Figure 2.5: Feature Diagram Editor - The tool allows to inspect the original documents, according to the features selected in the feature diagram.

vendors has been a time consuming task. Furthermore, during the definition of the model, we might have overseen some relevant details of the different solutions. These omissions might radically affect the feature model, and, consequently, invalidate the overall market analysis. The NLP-based approach presented in this section addresses these issues, by providing an automatic support for the selection of the commonalities and the variabilities, which can be exploited to define the global feature model.

2.2 NLP approach to Measuring Completeness

The starting point of a requirements definition process is very rarely a blank paper. More often, several input documents are placed on the desk of the requirements engineer, from legacy system documentation to reference standards, from transcripts of meetings with the customers to preliminary specifications. The content of these documents has to be taken into account when writing the requirements [55, 56], since it settles the background on which the future system can start to take its form. Such input documents are normally written in natural language (NL), and suitable natural language processing (NLP) tools can help identifying all the information that is relevant for the requirements. NLP approaches have been proposed in the past to identify significant abstractions that can aid the requirements process (e.g., [57, 58]). However, none of the existing approaches considers the *completeness* of the requirements with respect to the existing documentation. A requirements document that does not include the relevant information of the input documents - i.e., it is *incomplete* - could bring to several problems: if the missing information resides in the transcripts of meetings with the customers, the product might not address the customer's expectations; if some information is overseen from the reference standards, the resulting product might not comply to the norms; when concepts from legacy documentation and preliminary specifications are not taken into account, re-work on the product or on the process artifacts is hard to avoid.

Incomplete requirements specifications are often the cause of development costs overruns, project failures and even safety-critical accidents. To avoid these problems, the completeness of a requirements specification can be enforced by proper elicitation methods [59], by prototyping or by scenarios. All such methods aims at improving the completeness of the requirements with respect to the system-to-be, somehow foreseeing a possible implementation of the system to identify unexplored aspects. Instead, in this paper, we wish to focus on the completeness of the requirements w.r.t. the input documents of the requirements definition process, such as preliminary specifications, tran-

scripts of meetings, and reference standards.

In this section, we propose a NLP-based approach to *measure* and *improve* the completeness of a requirements specification with respect to the input documents of the requirements definition process.

A requirements document is *complete* with respect to the input documents if all the *relevant concepts* and *interactions* among concepts expressed in the input documents are also treated in the requirements. We refer to this type of completeness as *backward functional completeness*.

In order to measure such completeness, we provide two metrics that take into account the relevant terms and relevant relations among terms of the input documents. Furthermore, we provide a NLP approach to automatically extract such terms and relations. A prototype tool named Completeness Assistant for Requirements (CAR) has been developed, which suggests relevant information during the requirements definition phase, and automatically computes the degree of completeness of the requirements specification produced.

In particular the completeness of a requirements specification for the SYS-RS definition used in previous Section 1.6 in particular in Figure 1.9.

We evaluate the effectiveness of the approach with a pilot test, which is also used as a reference example in the remainder of the section. The pilot test concerns the definition of the requirements for an Automatic Train Supervision (ATS) component of a Communications-based Train Control system (CBTC). CBTC systems are introduced in chapter 1. These systems provide automatic train protection, train monitoring, and automated train driving. The ATS component of a CBTC is a centralized system that monitors and regulates the movement of the trains. The system automatically routes trains, and sends them speed profiles that shall be followed while moving through the railway network. It is normally equipped with a user interface where the ATS operator can view the position of all the trains, their schedule, and other information.

From the pilot test, we find that the CAR tool actually helps in improving the completeness of the requirements specification with respect to the input documents – in our case, the ATS reference standard. The tool suggests re-

lations about concepts that do not appear evident while reading the input document, and facilitates the identification of specific/alternative behaviours of the ATS system.

2.2.1 Defining and Measuring Completeness

In general, a requirements specification is complete if all the necessary requirements are included [60]. Several works have been presented in the literature to *define* and to *measure* the completeness of a requirements specification. In this paragraph, we review some definitions, which give a framework to understand the concept of *backward functional completeness* provided by the current section.

Completeness. A largely agreed definition of completeness of a requirements specification can be found in Boehm [61]. The definition states that a complete specification shall exhibit five properties: 1) No To-be-determined (TBD) items 2) No nonexistent references 3) No missing specification items (e.g., missing interface specifications) 4) No missing functions 5) No missing products (i.e., part of the actual software that are not mentioned in the specification).

Internal/External Completeness The definition is further conceptualized by Zowghi and Gervasi [62]. The first two properties defined by Bohem [61] are associated to *internal completeness*, and the second three properties to *external completeness*. Internal completeness can be measured by considering solely the information included in the specification. Instead, measuring external completeness requires additional information provided by domain experts, for example in the form of a domain model.

Feasible Semantic Completeness A more formal definition of external completeness - referred as *semantic completeness* - is given in Lindland et al. [63]. They look at the requirements specification as a *conceptual model* M, and they state that M has achieved semantic completeness if it contains all the statements about the domain D that are correct and relevant (i.e., $D \backslash M = \emptyset$). They observe that total semantic completeness cannot be achieved in practice, and they define the concept of *feasible semantic completeness* as $D \setminus M = S \neq \emptyset$.

The set S is composed of correct and relevant statements, but there is no statement in S such as the benefit of including it in the specification exceeds the drawback of including it.

Functional Completeness A further refinement of the concept, which goes toward the definition of a completeness measure, is provided by España et al. [64]. In line with the observations of Zowghi and Gervasi [62], the authors argue that, in order to compute the feasible semantic completeness, a reference model M_r shall be defined to conceptualize the domain D. By focusing on functional requirements, they consider the subset $FM_r \subset M_r$, which is a model of the functional requirements. Such a model is composed of functional encapsulations F_r, roughly "functions", and linked communications LC_r, roughly "messages". More formally, $F_r = FM_r \cup LC_r$.

A functional requirements specification FM shall be compared against this reference model FM_r to evaluate its completeness. Therefore, the specification FM shall be regarded as a composition of functional encapsulations F and linked communications LC (i.e., $FM = F \cup LC$). The introduced concepts are used to define two aspects of *functional completeness*:

- *functional encapsulation completeness:* all functional requirements specified in the reference model have been specified in the model (i.e., $F_r \setminus F = \emptyset$).
- *linked communication completeness:* all linked communications specified in the reference model have been specified in the model (i.e., $LC_r \setminus LC = \emptyset$).

In order to provide metrics associated to these aspects, the authors define the degree of functional encapsulation completeness as $degFEC = |F|/|F_r|$, and the degree of linked communication completeness as $degLCC = |LC|/|LC_r|$. In practice, computing these metrics requires the definition of a reference model for the functional requirements in terms of functions and linked communications.

2.2.2 Motivation

Besides the one applied by España et al. [64], several other measures for functional requirements completeness have been proposed in the literature (e.g., [65, 66, 67, 68, 69]). Nevertheless, the majority of such metrics deal with functional completeness defined with respect to the future implementation of the system[5]. Indeed, domain models [64], ontologies [69], identification of components [68], identification of system states [66], or expert analysis [65] are required to compute this kind of completeness. In other terms, domain experts are called to foresee a possible implementation of the system, possibly through a reference functional model FM_r. According to this vision, we refer to this kind of completeness as *forward functional completeness*. Instead, in our work we wish to focus on the completeness of the requirements with respect to the available input documents of the requirements definition process. The input documents might be transcripts of meeting with customers, preliminary specifications, reference implementation standards, or any other information specifically regarding the system under development. We refer to the completeness of a functional requirements specification with respect to the input documents as *backward functional completeness*.

Backward functional completeness is achieved by a functional requirements specification when (1) all the *relevant concepts* expressed in the input documents are treated in the requirements specification; (2) all the *relevant interactions* among concepts expressed in the input documents are treated in the requirements specification.

Consider for example the input document of our pilot test [2]. The document contains the sentence *"An ATS system shall have the capability to automatically track, maintain records of, and display on the ATS user interface the locations, [...], the train schedule and [...]"*. Besides the other content, such a sentence tells that the ATS user interface is supposed to display the schedule of the trains. Therefore, the requirement specification is expected to include the concepts of "ATS user interface" and "train schedule". Furthermore, require-

[5]One exception is [67], where completeness is evaluated against higher-level requirements

ments shall be provided that define the *interaction* among the two concepts (i.e., the fact that the ATS user interface shall display the train schedule).

Achieving backward functional completeness ensures that no relevant information contained in the input documents is left out from the specification. Measuring this type of completeness can give higher confidence on the quality of the specification. Therefore, a metric is required to measure this kind of completeness. Furthermore, we are also interested in establishing whether a positive correlation holds between such completeness and the completeness of the specification with respect to the system to be (i.e., the *forward functional completeness*).

Bearing these observations in mind, we define three research questions, which are addressed by the current section: **RQ1.** How to *measure* the backward functional completeness of a requirements specification document? **RQ2.** How to *improve* the backward functional completeness of a requirements specification document? **RQ3.** Does the backward functional completeness help in improving the *forward functional completeness* of the specification?

The first question is answered by computing two completeness metrics that consider the number of relevant terms that are used in the input documents, and the number of relevant relations among terms (Sect. 2.2.3). Roughly, a document is more complete than another if more relevant terms and more relevant relations are included in the document. The second question is answered through a prototype tool that suggests relevant terms to be included in the requirements, and that considers the relations among terms (Sect. 2.2.6). The third question is answered through a pilot test, where we have evaluated the forward functional completeness of the requirements produced with the proposed tool, and without the proposed tool (Sect. 2.2.7).

2.2.3 Metrics for Backward Functional Completeness

Measuring the backward functional completeness of a requirements specification requires the definition of specific metrics (**Research Question 1**). Here, we define two metrics. The first one, named *degree of concept completeness*,

measures how many relevant concepts that are expressed in the input documents are treated also in the specification. The second one, named *degree of interaction completeness*, measures how many relevant interactions that are expressed in the input documents are treated also in the specification.

More formally, we define the two metrics as follows. Let T be the set of relevant concepts expressed in the input documents, and let $Q \subseteq T$ be the set of such concepts expressed in the requirements specification. We define the *degree of concept completeness* of a requirements document $\mathcal{D}$ with respect to a set of input documents $\mathcal{I}$ as $degCC(\mathcal{D}, \mathcal{I}) = |Q|/|T|$.

Now, let U be the set of relevant interactions among concepts expressed in the input documents, and let $R \subseteq U$ be the set of relevant interactions among concepts expressed in the requirements specification. We define the *degree of interaction completeness* of a requirements document $\mathcal{D}$ with respect to a set of input documents $\mathcal{I}$ as $degIC(\mathcal{D}, \mathcal{I}) = |R|/|U|$.

Given a requirements document and the corresponding input documents, we would like to compute the two metrics in an automated manner.

We argue that the relevant concepts expressed in the input documents can be approximated with the relevant *terms* included in such documents. Furthermore, relevant interactions among concepts can be approximated with the relevant *relations* among terms. Therefore, we define a NLP approach to automatically identify relevant terms and relations among terms in the input documents.

2.2.4 Identification of Relevant Terms

The proposed method for the identification of relevant terms is based on a novel natural language processing approach, named *contrastive analysis* [50], for the extraction of *domain-specific terms* from natural language documents. In this context, a *term* is a conceptually independent linguistic unit, which can be composed by a single word or by multiple words. For example, consider the document that we have used in our pilot test [2]. In such document, "Automatic Train Supervision" is a term, while "Supervision" is not a term, since in

the textual documents considered in our study it often appears coupled with the same words (i.e., "train", "route"), and therefore it cannot be considered as conceptually independent.

The *contrastive analysis* technology aims at detecting those terms in a document that are *specific* for the domain of the document under consideration [50, 53]. Roughly, contrastive analysis considers the terms extracted from domain-generic documents (e.g., newspapers), and the terms extracted from the domain-specific document to be analysed. If a term in the domain-specific document highly occurs also in the domain-generic documents, such a term is considered as domain-generic. On the other hand, if the term is not frequent in the domain-generic documents, the term is considered as domain-specific.

In our work, the documents from which we want to extract domain-specific terms are the input documents of the requirements definition phase. The proposed method requires two steps. First, conceptually independent expressions (i.e., *terms*) are identified (*Identification of Terms*). Then, *Contrastive Analysis* is applied to select the terms that are domain-specific. *Identification of Terms* and *Contrastive Analysis* are introduced in Section 2.1.2.

Consider again our pilot test. After the contrastive analysis, a term such as "train" – which is highly frequent in the document (57 occurrences), but is also frequent in the contrastive corpora – is ranked lower than "ATS user interface". Indeed, this term has 8 occurrences in the document, but is uncommon in the contrastive corpora.

After the contrastive analysis, we have a list of terms, together with their ranking according the function $TRank$ ($TRank = R_i$ see Section 2.1.2.2), and their frequency in I. The more a term is likely to be domain-specific, the higher the ranking. From the list, we select the terms that received the higher ranking. The choice shall be made according to a *domain relevance threshold* τ. If $TRank(t) \geq \tau$ the term will be selected as *relevant*. The value of τ is defined over normalized values, where the rank of each term is divided by the maximum value of $TRank$. The selection of τ shall be performed by a domain expert after reviewing the lists of terms extracted. Normally, a value of $\tau = 0.99$ allows selecting most of the relevant terms.

Assuming that the set of selected terms $\bar{T}$ provides an approximation of the relevant concepts of the input documents T, we can approximate the degree of concept completeness as $degCC(\mathcal{D}, \mathcal{I}) \approx |\bar{Q}|/|\bar{T}|$, where $\bar{T} = \{t \subset I : TRank(t) \geq \tau\}$, and $\bar{Q} = \mathcal{D} \cap \bar{T}$. For example, in our case study, we have $|\bar{T}| = 67$ relevant terms extracted from the input documents (see Table 2.1 for examples). In the first experiment, the requirements produced by subject A included $|\bar{Q}| = 46$ of such terms. Therefore $degCC(\mathcal{D}, \mathcal{I}) \approx 68.7\%$.

2.2.5 Identification of Relevant Relations

In order to identify relevant relations among terms, we first select all the terms t extracted in the previous step, regardless of their ranking. Then, we search for possible relations among such terms. We state that there is a relation $u = (t_j, t_h)$ between two terms t_j, t_h if such terms appear in the same sentence or in neighboring sentences. In our case, we select the previous and the following sentence. In order to give a rank to such relation, we use the Log-likelihood metric for binomial distributions as defined in [70]. The explanation of such metric is beyond the scope of this section. Here, we give an idea of the spirit of the metric. Roughly, a relation holds between two terms if such terms frequently appear together. Moreover, the relation is stronger if the two terms do not often occur with other terms. In other words, there is a sort of *exclusive relation* among the two terms. For each couple of terms t_j, t_h occurring in neighboring sentences of the input document I, we associate a rank according to the Log-likelihood metric, which represents the degree of their relation $u = (t_j, t_h)$:

$$RRank(u) = \text{Log-likelihood}(t_j, t_h)$$

In our pilot test, the term "re-routing of trains" has a relation with "movement of trains" and with "ATS user interface". However, the relation is stronger (i.e., more exclusive) with the former ($RRank = 14.88$ *vs* $RRank = 8.85$), since the latter often occurs with other terms. Indeed, the ATS user interface is required to show several information, besides those concerning re-routing of

the trains.

After this analysis, we have a list of relations, together with their ranking according to the function $RRank$. From the list, we select the terms that received the higher ranking. The choice shall be made according to a *relation degree threshold* ρ. If $RRank(u) \geq \rho$, the relation will be selected as *relevant*. The selection of ρ shall be performed by a domain expert after reviewing the lists of relations extracted with the proposed method. Normally, a Log-likelihood above 10.83 is recommended to select only relevant relations. However, lower thresholds can be chosen, if more relations are required.

Assuming that the set of selected relations $\bar{U}$ provides an approximation of the relevant interactions U in the input documents, we can approximate the degree of interaction completeness as $degIC(\mathcal{D}, I) \approx |\bar{R}|/|\bar{U}|$, where $\bar{U} = \{u \in \bar{T} \times \bar{T} : RRank(u) \geq \rho\}$, and $\bar{Q} = (\mathcal{D} \times \mathcal{D}) \cap \bar{U}$. For example, in our case study, we have $|\bar{U}| = 316$ relations extracted from the input documents (see Table 2.2 for examples). In the first experiment, the requirements produced by subject A included $|\bar{R}| = 54$ of such relations. Therefore $degIC(\mathcal{D}, I) \approx 17.1\%$.

2.2.6 A Word-game to Support Requirements Definition

We would like to provide means to improve the backward functional completeness of a requirements specification (**Research Question 2**). We argue that the backward functional completeness of a requirements specification is normally hampered by two problems: (1) *missing concepts:* the person who writes the requirements might forget to consider relevant concepts of the problem, either because she postpones their analysis, or because they are unclear and hard to specify, or because the input documents include too many concepts to consider them all; (2) *missing concept interaction:* when one writes a requirement, she might be concentrated on the specific function that she is defining, and oversee possible interactions among elements.

We have implemented a prototype tool named Completeness Assistant for Requirements (CAR), which addresses these problems by automatically suggesting possible relevant terms and possible relevant relations among

terms to be used in the requirements. The relevant terms and relations are extracted from the input documents (e.g., transcripts of meeting with the customers, reference standards, preliminary requirements) according to the approach explained in Sect. 2.2.3. Therefore, the tool starts with a set $\bar{T}$ of relevant terms, and a set $\bar{U}$ of relevant relations. Furthermore, the degree of concept completeness and the degree of interaction completeness is computed at run-time while the requirements manager writes down the requirements.

Fig. 2.6 shows the interface of CAR. The figure is used as a reference example to explain the working principles of the tool. The example, adapted from our pilot test, concerns the definition of the requirements for an Automatic Train Supervision (ATS) system. An ATS system is indroduced in chapter 1. The input document, in the example, is a reference international standard [2], which is used as a starting point to write the requirements for the ATS system. In general, the tool can work with any kind of natural language input document, such as interviews, transcripts of meetings with the customers, etc.

The tool is a sort of word-game. The main steps of the game are summarized below:

1. The tool suggests to write a requirements with three terms. The first term (conductor , in Fig. 2.6) is extracted from the set of relevant terms, while the other two terms (control , train doors) are extracted from the set of relevant relations. The three terms are also highlighted in the original document, which is loaded to the bottom frame of the interface. In the current version of the tool, the extraction is random. Nevertheless, smarter approaches can be devised that choose the terms by taking into account their relevance, their position, or the previously written requirement.

2. The user writes a requirement, possibly using the three terms suggested. An example requirement that employs the three terms is *"The ATS system shall notify the inhibition of* control *of the* train doors *to the train* conductor*"*. Then, the user adds the requirement to the central panel by

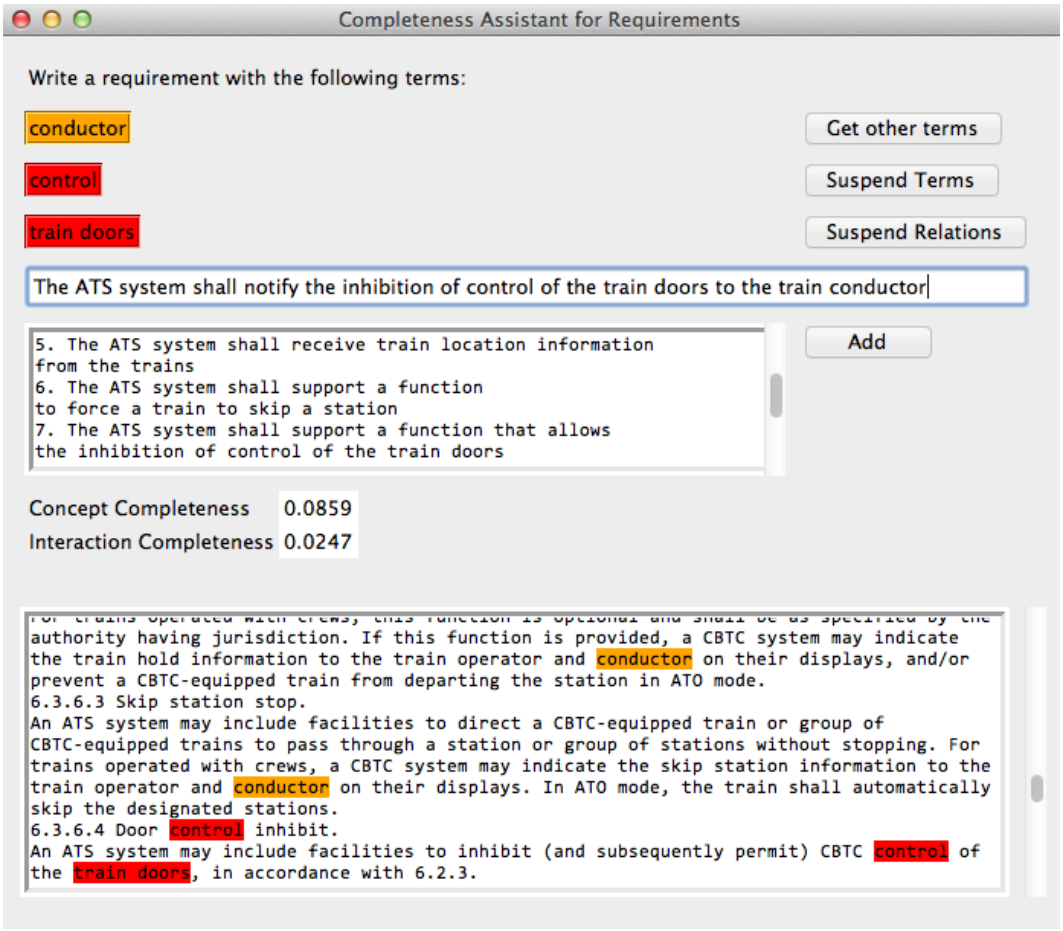


Figure 2.6: User interface of the tool.

pressing the button **Add**. It is worth noting that a requirement like the one presented above could not be deduced by simply reading the text of the input document. It is actually an additional behaviour inspired by the suggested terms. Indeed, a relation between the "conductor" and the "train doors" was not specified in the original input document, as one can see from the fragment displayed in Fig. 2.6.

3. The system checks if the user used any relevant term or relevant relations, and consequently increases the degree of **Concept Completeness** and the degree of **Interaction Completeness**. These values are computed as $|\bar{Q}|/|\bar{T}|$ and $|\bar{R}|/|\bar{U}|$, respectively, as explained in Sect. 2.2.4 and 2.2.5.

When relevant concepts are found within the requirement, these are added to the set $\bar{Q}$. When relevant relations are found, these are added to the set $\bar{R}$. The current values of the metrics are shown below the panel that lists the requirements.

4. The system automatically suggest other terms to be used in the following requirement.

If a relevant term or relation is suggested twice, and the user does not employ it in the requirement, such term/relation is marked as *not relevant*. Therefore, the completeness scores are adjusted consequently (i.e., $|\bar{T}|$ or $|\bar{U}|$ are decreased).

In some cases, the user might not be interested in writing a requirement that includes all the suggested terms. In other cases, the user might want to focus on the suggested terms/relations to write more than one requirement. With the normal behaviour of the tool, new terms/relations would be automatically suggested in these cases after pressing the button **Add**. As explained, if such terms/relations are not used, they are marked as *not relevant*, and will not be presented anymore among the suggestions. Therefore, we added the **Suspend Terms** and **Suspend Relations** buttons, to suspend the automated suggestion of terms and relations, and prevent the tool from marking them as not relevant.

If new relations among terms are reported in a requirement, these new relations shall be added to the relevant relations $\bar{U}$. In our case, the relations between "conductor" and the other two terms are added to $\bar{U}$. Similarly, if some terms are used that were not identified as relevant in the initial analysis, such terms shall be stored among the relevant terms $\bar{T}$. These situations do not influence the computation of the backward completeness (also $|\bar{Q}|$ and $|\bar{R}|$ increase like $|\bar{T}|$ and $|\bar{U}|$). Nevertheless, we argue that storing and reviewing the new concepts and relations can help understanding if the requirements specification provides additional information with respect to the input documents.

2.2.7 Pilot Test

We have performed a pilot test to assess the effectiveness of the proposed approach, and to evaluate the correlation between the backward functional completeness and the forward functional completeness (**Research Question 3**) of a requirements specification.

In the pilot test, referred as subject A and subject B, were required to write requirements for an ATS system, according to the generic requirements provided by the standard IEEE Std 1474.1-2004 [2].

The requirements have been written with the support of the tool, and without the support of the tool. The goal was to compare the degree of backward functional completeness and the degree of forward functional completeness achieved in the two cases.

More specifically, the pilot test required four steps, which are described below.

1. Input document reading: the chapter concerning the ATS of the IEEE Std 1474.1-2004 [2] - about 5 pages long - was used as input document for the requirements definition task. Subject A and B were asked to read the input document to have a first understanding of the general needs of the system.

2. Tool set-up: from the input document, 67 relevant terms and 316 relevant relations have been automatically extracted. To this end, a threshold of 99% and a threshold of 10 were chosen as domain relevance threshold τ, and relation degree threshold ρ, respectively. In Table 2.1 and 2.2, we provide representative examples of relevant terms and relevant relations extracted from the document. These terms and relations have been fed into the tool to support the definition of the requirements.

3. Requirements definition Phase 1: subject A and B were asked to write the requirements. Subject A operated with the support of the tool, and subject B operated without the tool. The requirements definition lasted one hour.

4. Requirements definition Phase 2: subject A and B were asked again to write the requirements. Subject B operated with the tool, and subject A operated without the tool until they produced the same amount of requirements

produced in the previous step (i.e., if a subject produced n requirements in Phase 1, he should have produced n requirements also in the Phase 2). Given a subject, this choice allows comparing the completeness scores achieved in the two phases on the same amount of requirements.

The subjects chosen for the test were involved in the definition of the principles of CAR, while the approach for term/relation extraction was defined and implemented by the second subject only. Therefore, we argue that the expectations of the two test subjects on the success of the solution had a limited influence on the result of the test. Indeed, they did not know which types of terms/relations would be considered relevant by the tool, and could not influence the test by avoiding the usage of relevant terms/relations when the tool was not used. This is especially true for Subject B, who performed his first experiment without the tool. But it is also true for Subject A, since during the first experiment he viewed only a limited part of the terms/relations extracted by the tool (i.e., the suggested terms/relations).

Term	**TRank (%)**	**Freqency**
CBTC	100.0	44
ATS	$99.99999 + 0.99769 \times 10^{-6}$	43
ATS system	$99.99999 + 0.8456 \times 10^{-6}$	19
ATS user interface	$99.99999 + 0.29614 \times 10^{-6}$	8
train location	$99.99999 + 0.1231 \times 10^{-6}$	7
train	$99.99999 + 0.1185 \times 10^{-6}$	57
conductor	$99.99997 + 0.73215 \times 10^{-6}$	8
station	$99.99979 + 0.57378 \times 10^{-6}$	12

Table 2.1: Examples of relevant terms

2.2.8 Quantitative Evaluation

We evaluated the results of the test by computing the backward functional completeness of the produced requirements for the two subjects. Then, we computed the forward functional completeness according to the metrics provided by España et al. [64]. The degree of functional encapsulation completeness $degFEC$, and the degree of linked communication completeness $degLCC$ require the definition of a reference model for the system. In our case, we have

Relation	**RRank**	**Freqency**
(conductor, ATS system)	35.1402383629	6
(ATS user interface, position of trains)	17.9938334306	2
(station, train at station)	16.1777267317	2
(speed regulation function, service brake rates)	14.8834871304	1
(train fault reporting, train health data)	14.8834871304	1
(re-routing of trains, movement of trains)	14.8834871304	1
(equipment, supplier)	13.1023727742	2
(ATS user interface, movement authorities)	12.4872415276	2
(station departure time, train service)	12.1108984081	1

Table 2.2: Examples of relevant relations

employed a preliminary system specification where functions and linked communications were listed. The reference model defines 21 functions and 10 linked communications for the ATS system. The document was edited in the context of the Trace-IT project (see). It is worth noting that the reference model was provided *before* the definition of the method presented in this section. Table 2.3 summarizes the results of the test.

Subject	Num. Reqs	Tool	$degCC$	$degIC$	$degFEC$	$degLCC$
A	36	Yes	68.7%	17.1%	47.6%	40%
		No	52.3%	12.8%	61.9%	50%
B	21	Yes	67.2%	24.5%	47.6%	50%
		No	58.2%	11.6%	33.3 %	50%

Table 2.3: Results of the pilot tests

Backward functional completeness. We see that, for both subjects, the backward functional completeness, estimated with $degCC$ and $degIC$, is higher when the tool is employed ($\Delta degCC = 12.7\%$ and $\Delta degIC = 8.6\%$ in average). Therefore, in our pilot test, the usage of the tool actually helped in improving the backward functional completeness of the requirements specification. Further-

more, we argue that if a larger amount of input documents would be employed, the benefit given by the usage of the tool would be even more evident. The CAR tool helps in the navigation of the input documents. Without tool support, coherent navigation would be hardly practicable in the case of many documents. Moreover, with a larger amount of information, the statistics that bring to the set of relevant terms/relations would be more accurate, and the consequent suggestions given by the tool would be more meaningful.

Forward Functional Completeness. Conflicting results have been found concerning the effectiveness of the approach with respect to forward functional completeness, estimated through $degFEC$ and $degLCC$. Indeed, we see that subject A achieved a lower value for both metrics when using the tool with respect to the values obtained when the tool was not employed ($\Delta degFEC = -14.3\%$, $\Delta degLCC = -10\%$). Instead, subject B achieved a higher value for $degFEC$ when using the tool ($\Delta degFEC = 14.3\%$), while equivalent values for $degLCC$ were obtained in Phase 1 and 2. Therefore, from our test, we cannot identify a positive correlation between the degree of backward functional completeness and the degree of forward functional completeness. Instead, we argue that the results obtained might be related to the *order* that was followed by the two subjects in performing the tasks. Subject A performed the experiment with CAR *before* writing the requirements without the tool, while for subject B was the other way around. Both subjects achieved a higher degree of completeness during Phase 2. Basically, a higher degree of completeness was obtained when the subjects acquired a higher confidence with the topic of the requirements, since they already defined requirements for the system in Phase 1.

2.2.9 Qualitative Evaluation

We have performed a qualitative analysis of the produced requirements to understand which were the main differences between the requirements produced with CAR and those produced without the tool. Interesting results have been found. We have identified two main differences: 1) requirements produced

with CAR tend to be more specific, while requirements produced without the tool are more high-level; 2) requirements produced with CAR tend to identify alternative behaviors of the system. Representative examples of requirements produced *without* the support of the tool by subject A are:

- R_1. *The ATS system shall send the desired speed profile to the trains*
- R_2. *The ATS system shall have the capability to define temporary speed restrictions for the trains*
- R_3. *The ATS system shall implement the functionality of train routing*

These requirements are quite generic, and do not add too much content compared to the input document. Instead, more specific requirements are produced with the tool. For example, the following requirement was produced when the tool suggested the term "emergency brake application" and the relations <"response", "wet rail">:

- R_4. *The ATS system shall adjust the speed profile of the trains in* response *to* wet rail *conditions in order to avoid* emergency brake application.

Such requirement can be regarded as a specialization of R_1 and R_2, since it explains the specific condition (i.e., the wet rail) that requires temporary speed restrictions. The following requirement is an example of an alternative behavior identified with the support of the tool. In this case, the relations suggested was <"re-routing", "service disruptions">:

- R_5. *The ATS system shall be capable of supporting* re-routing *of trains in response to* service disruptions.

This requirement shows an alternative behavior (i.e., re-routing) of the routing functionality identified by requirement R_3. According to this preliminary analysis, we argue that the proposed tool can play a complementary role during requirements definition. Indeed, it can be used as a support tool to identify specific cases, and alternative behaviors that tend to be overseen in requirements definition approaches based solely on the analysis of the input documents.

Chapter 3

Development of a sub-component within Formal Methods

In CBTC platforms, a prominent role is played by the Automatic Train Supervision (ATS) system, which automatically dispatches and routes trains within the metro network. In absence of delays, the ATS coordinates the movements of the trains by adhering to the planned timetable. In presence of delays, the ATS has to provide proper scheduling choices to guarantee a continuous service and ensure that each train reaches its destination. In particular, this implies that the ATS shall necessarily avoid the occurrence of *deadlock* situations, i.e., situations where a group of trains block each other, preventing in this way the completion of their missions. After studying this scenario of the ATS system, we have decided to apply the Formal Methods to manage *degraded modes operation*, in the development of an ATS system. This choice stems from the need to manage situations that would have great impact on the operation of the railway system, with the risk of blocking the railway network, even if guaranteeing safety.

Formal methods have been widely and successfully used in the railway con-

text [71], but usually they are applied only to their safety critical components.

This Chapter present the approach used in the design of the scheduling kernel of an Automatic Train Supervision (ATS) system. A formal model of the railway layout and of the expected service has been used to identify all the possible critical sections of the railway layout in which a deadlock might occur. For each critical section, the prevention of the occurrence of deadlocks is achieved by constraining the set of trains allowed to occupy these sections at the same time. The identification of the critical sections and the verification of the correctness of the logic used by the ATS is carried out by exploiting a model checking verification framework locally developed at ISTI-CNR and based on the tool UMC [72].

3.1 Formal Methods in ATS

This chapter presents the experience in the design of the scheduling kernel of an ATS system. A prototype of the ATS system has been implemented, which operates on a simple but not trivial metro layout with realistic train missions. To address the problem of deadlock avoidance in our ATS prototype, we have decided to develop sound solutions based on formal methods. In a short preliminary work [73], we have outlined a model-checking approach for the problem of deadlock avoidance. Such an approach included several manual steps, and did not consider the presence of *false positives* (i.e., cases in which a train is unnecessarily disallowed to proceed). Furthermore, the current strategy exploits the usage of model checking also to address the problem of false positives.

The ATS that we have designed prevents the occurrence of deadlocks by performing a set of runtime checks just before allowing a train to move further. The set of checks to be performed is retrieved from statically generated configuration data that are validated by means of model checking. Our approach to produce valid configuration data starts with the automatic identification of a set of *basic cases* of deadlocks. This goal is achieved by statically analysing

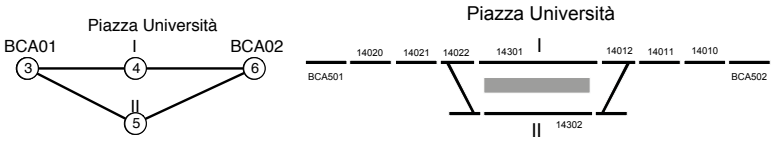
the missions of all the trains, and providing a set of preliminary constraints that can be used to address the basic cases of deadlocks. Then, we build a formal model of the scheduling kernel of the ATS that includes the constraints associated to the basic cases of deadlock. We use such a formal model to verify the absence of *complex cases* of deadlocks, and to assess the absence of false positive cases. To this end, we apply model checking by means of the UMC (UML Model Checker) tool, which is a verification environment working on UML-like state machines [72]. When complex cases of deadlock are found, the formal model is updated with additional checks to address such cases. The validation process iterates until the ATS configuration data are proven to avoid all possible cases of deadlocks. The verification of the configuration data for the full railway yard is performed by decomposing it into multiple regions to be analysed separately, and by proving that the adopted decomposition allows extending the results to the full layout.

The chapter is structured as follows. In Sect. 3.2, we illustrate an abstract model of the ATS, together with the metro layout and the missions of our ATS prototype. In Sect. 3.3, the basic cases of deadlocks are described, and the approach to identify and automatically avoid such cases is outlined. Sect. 3.4 explains how complex cases of deadlocks can occur, and introduces the problem of false positives. Sect. 3.5 describes the formal model provided for the ATS and the approach adopted to verify the absence of deadlocks and false positives. In Sect. 3.6, we describe how we have partitioned the full layout.

3.2 An Abstract Model of the System

The abstract behavior of the kernel of the ATS system can be seen as a state machine. This state machine has a local status recording the current progress of the train missions and makes the possible scheduling choices among the trains which are allowed to proceed.

Train movements can be observed and modeled at different levels of abstractions. In Figure 3.1 we show two levels of abstraction of the train movement,



(a) Itinerary level view (b) Track circuit level view

Figure 3.1: The itinerary and track circuit level view of a station

namely the itinerary level view and the track circuit level view. An *itinerary* is constituted by the sequence of track circuits (i.e., independent line segments) that must be traversed for arriving to a station platform from an external entry point, or for leaving from a station platform towards an external exit point. The itinerary are composed in this way to preserve resources. Track circuits are not visible at the itinerary level view, which is our level of observation of the system for the deadlock-avoidance problem. We assume that the train is contained in a single track circuit. Instead, at the interlocking management level, we would be interested in the more detailed track circuit level view, because we have to deal with the setting of signals and commutation of switches for the preparation of the requested itineraries. Notice that it is the task of the interlocking system (IXL) to ensure the safety of the system by preparing and allocating a requested itinerary to a specific train. At the ATS level it is just a performance issue the need to avoid the issuing of requests which would be denied be the IXL, or to avoid sequences of safe (in the sense risk free) train movements but which would disrupt the overall service because of deadlocks.

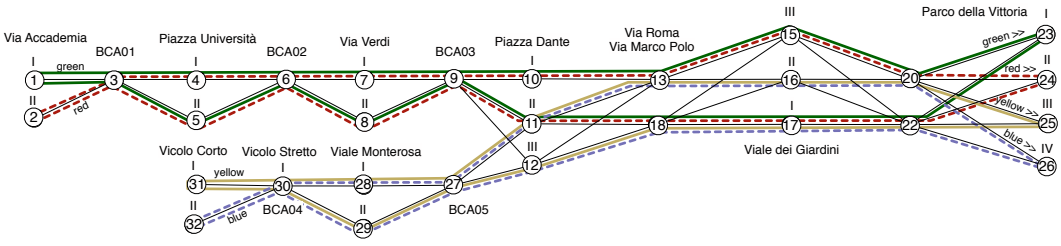


Figure 3.2: The yard layout and the missions for the trains of the green, red, yellow and blue lines

In our case, the overall map of the railway yard which describes the various interconnected station platforms and station exit/entry points (itinerary endpoints) is shown in Figure 3.2. Given our map, the mission of a train can be seen as a sequence of itinerary endpoints. In particular, the service is constituted by eight trains which cyclically start their missions at the extreme points of the layout, traverse the whole layout in one direction and then return to their original departure point. The missions of the eight trains providing the green/red/yellow/blue line services shown in Figure 3.2, are represented by the data in Table 3.1.

```
Green1: [1,3,4,6,7,9,10,13,15,20,23,22,17,18,11,9,8,6,5,3,1]
Green2: [23,22,17,18,11,9,8,6,5,3,1,3,4,6,7,9,10,13,15,20,23]
Red1:   [2,3,4,6,7,8,9,10,13,15,20,24,22,17,18,11,9,8,6,5,3,2]
Red2:  [24,22,17,18,11,9,8,6,5,3,2,3,4,6,7,8,9,10,13,15,20,24]
Yellow1:[31,30,28,27,11,13,16,20,25,22,18,12,27,29,30,31]
Yellow2:[25,22,18,12,27,29,30,31,30,28,27,11,13,16,20,25]
Blue1:  [32,30,28,27,11,13,16,20,26,22,18,12,27,29,30,32]
Blue2:  [26,22,18,12,27,29,30,32,30,28,27,11,13,16,20,26]
```

Table 3.1: The data for the missions

In absence of deadlock avoidance checks, in our abstract model, trains are allowed to move from one point to the next under the unique condition that the destination point is not assigned to another train. This transition is modeled as an atomic transition, and only one train can move at each step. We are interested in evaluating the traffic under any possible condition of train delays. Therefore we abstract completely away from any notion of time and from the details of the time schedules. Indeed, if we consider all the possible train delays, the actually planned times of the time table become not relevant.

3.3 The basic cases of deadlock

A *basic deadlock* occurs when we have a set of trains (each one occupying a point of the layout) waiting to move to a next point that is already occupied by another train of the set. In our railway scenario this means that we have

a basic deadlock when two trains are trying to take the same itinerary in opposite directions, or when a set of trains are moving around a ring which is completely saturated by the trains themselves. We consider as another case of *basic deadlock* the situation in which two trains are trying to take the same linear sequence of itineraries in opposite directions.

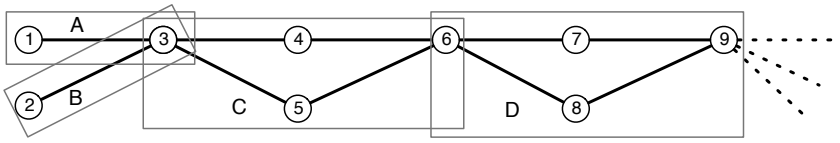


Figure 3.3: A sample selection of four basic critical sections from the full layout

For example, if we look at the top-left side of our yard layout we can easily recognize four of these zones in which the four Green and Red trains might create one of these basic deadlocks (see also Figure 3.3):

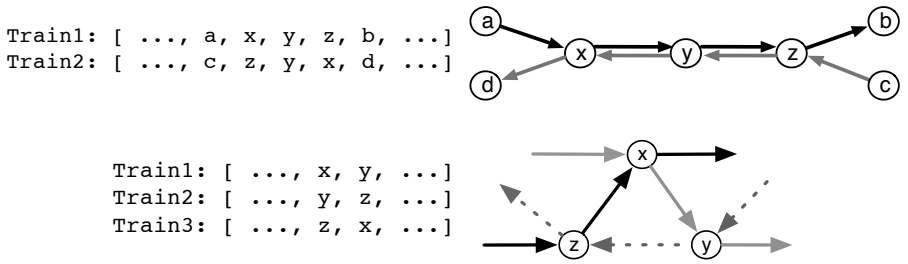
a) The zone A `[1-3]` when occupied by `Green1` and `Green2`.

b) The zone B `[2-3]` when occupied by `Red1` and `Red2`.

c) The zone C `[3-4-6-5]` when occupied by the four Green and Red trains.

d) The zone D `[6-7-9-8]` when occupied by the four Green and Red trains.

The first step of our approach to the deadlock free scheduling of trains consists in statically identifying all those zones of the railway layout in which a basic deadlock might occur. We call these zones *basic critical sections*. We have already seen the two basic kinds of critical sections, namely *ring sections* and *linear sections*, which are associated to the basic forms of deadlocks mentioned before. Given a set of running trains and their missions, the set of basic sections of a layout are statically and automatically discovered by comparing the various missions of all trains. In particular, linear sections are found by comparing all possible pairs of train missions. For example if we have that:
then (x,y,z) constitutes a basic linear section of the layout. Similarly if we have for example three trains such that:



then (x,y,z) constitutes a basic ring section of the layout.

The next step of our approach consists in associating one or more counters to the critical sections in order to monitor at execution time that the access to them will not result in a deadlock. It is indeed evident that if we allow at most N-1 trains to occupy a ring section of size N no deadlock can occur on that ring. Similarly, in the case of linear sections, we could use two counters (each one counting the trains moving in one direction) and make sure that one train enters the section only if there are no trains coming from the opposite side (while still allowing several trains to enter the section from the same side). When a train is allowed to enter a basic critical section the appropriate counter is increased; when the train is no longer a risk for deadlocks (e.g. moves to an exit point of the section) the counter is decreased.

The above policy can be directly encoded in the description of the train missions by associating to each itinerary endpoint the information on which operations on the counters associated to the entered/exited sections should be performed when moving to that endpoint. We call the description of the train missions extended with this kind of information *extended train mission*.

Let S be a generic name of a section. In the following we will use the notation S+ to indicate that a train is reaching an entry point of a ring section S, (correspondingly increasing its counter), and the notation S- to indicate that a train is reaching an exit point of section S (correspondingly decreasing its counter). The notation SR+ (SR-) indicates that a train is reaching the entry (exit) point of a linear section S from when arriving from its right side. The notation SL+ (SL-) indicates that a train is reaching the entry (exit) point of

a linear section S from when arriving from its left side.

The cases of deadlocks over the basic sections shown in Figure 3.3 can be avoided by extending the missions of the trains of the green and red lines (originally shown in Table 3.1) in the following way:

```
Sections:
   [A, B, C max 3, D max 3]
Train Missions:
   Green1: [(AL+)1,(AL-,C+)3,4,(C-,D+)6,7,(D-)9,10,13,15,20,23,
              22,17,18,11,(D+)9, 8,(D-,C+)6,5,(C-,AR+)3,(AR-)1 ]
   Green2: [23,22,17,18,11,(D+)9,8,(D-,C+)6,5,(C-,AL+,AR+)3,(AR-)1,
            (AL-,C+)3,4,(C-,D+)6,7,(D-)9,10,13,15,20,23 ]
   Red1: [(BL+)2,(BL-,C+)3,4,(C-,D+)6,7,(D-)9,10,13,15,20,24,
           17,18,11,(D+)9,8,(D-,C+)6,5,(C-,BR+)3,2 ]
   Red2: [24,22,17,18,11,(D+)9,8,(D-,C+)6,5,(C-,BL+,BR+)3,(BR-)2,
            (BL-,C+)3,4,(C-,D+)6,7,(D-)9,10,13,15,20,24 ]
```

Given the discovered set of basic sections, the description of the extended train missions for all running trains can be automatically computed without effort. By performing such an initial static analysis on the overall service provided by our eight train missions shown in Figure 3.2 we can find eleven basic critical sections (see Figure 3.4) and automatically generate the corresponding extended mission descriptions for all trains. All this automatically generated data about critical sections and extended missions will be further analyzed and validated before being finally encoded as ATS configuration data and used by the ATS to perform at runtime the correct train scheduling choices.

3.4 From basic to composite sections

Our set of basic critical sections actually becomes a new kind of resource shared among the trains. When moving from one section to another, a train may have to release one section and acquire the next one. Again, this behavior can be subject to deadlock. Let's consider the example of regions A, B, C shown in Figure 3.5.

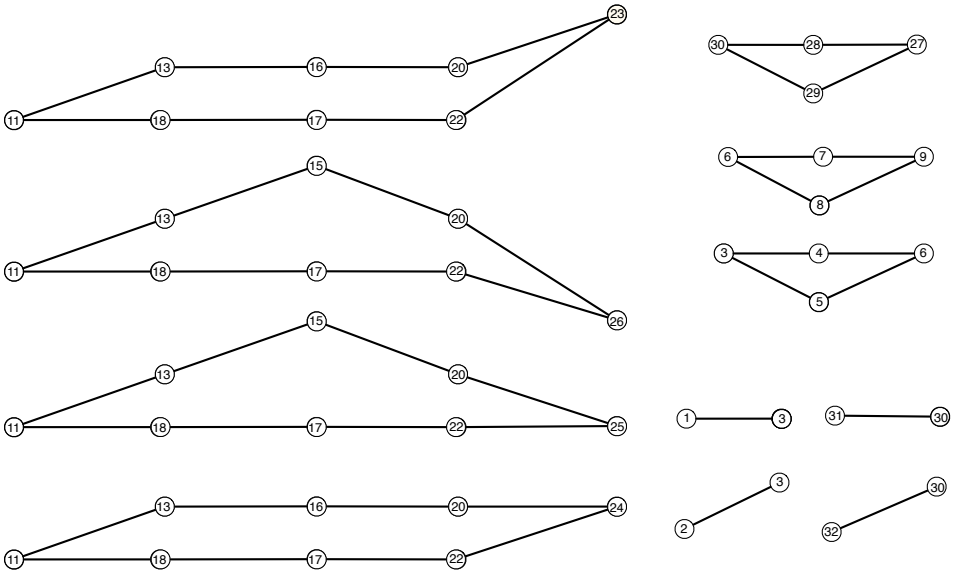


Figure 3.4: All the basic critical sections of the overall layout

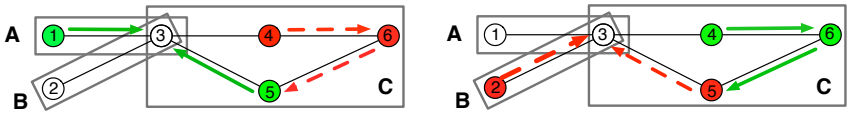
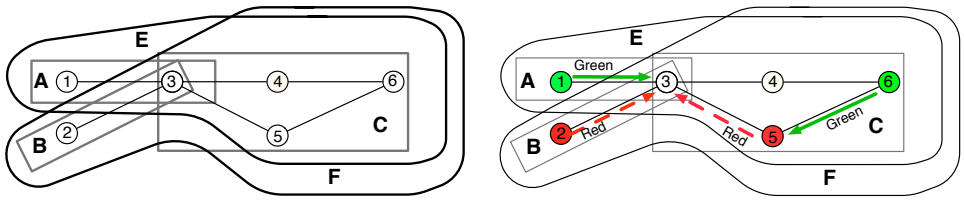


Figure 3.5: Deadlock situations over the composition of basic critical sections

In the left case (the right case is just an analogous example), train `Green2` cannot exit from critical section C because it is not allowed to enter critical section A. Moreover, train `Green1` is not allowed to leave critical section A because it is not allowed to enter critical section C. The deadlock situation that is generated in the above case is not a new case of deadlock introduced by our deadlock avoidance mechanism, but just an anticipation of an unavoidable future deadlock (of the basic kind) which would occur if we allow one of the two trains to proceed. To solve these situations, we can introduce two additional composite critical section E and F respectively over the points `[1-3-4-6-5]`, (section A plus section C) and `[2-3-4-6-5]` (section B plus section C), which are allowed to contain at most three of the trains `Green1, Green2, Red1, Red2)`. These new sections are shown in Figure 3.6a. The missions of the Green and Red trains are correspondingly updated to take into consideration also these new sections.



(a) The composite sections E and F

(b) A potential deadlock caused by false positives

Figure 3.6: Composite sections and new deadlock case

It is very important that our mechanism does not give raise to false positive situations, i.e, situations in which a train is unnecessarily disallowed to proceed. False positive situations, in fact, not only decrease the efficiency of the scheduling but also risk to propagate to wider composite sections, creating even further cases of false positives or deadlocks.

Let us consider the situation shown in Figure 3.6b. The red train in `2` is not allowed to proceed in point `3` because section E already contains its maximum of three trains. The same occurs for the green train in point `1` (section F already has three trains). As a consequence nobody can progress, while, on

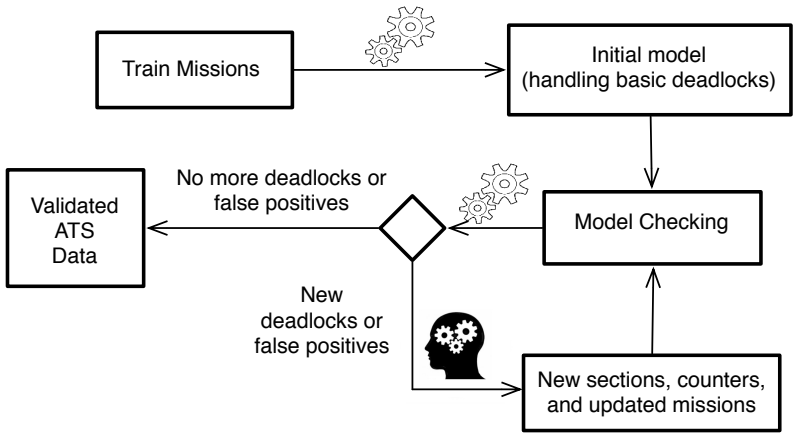


Figure 3.7: The ATS configuration data validation process

the contrary, nothing bad would occur if the red train in 2 was allowed to proceed.

As we build greater composite sections it becomes extremely difficult to manually analyze the possible effects of the choices. We need a mechanical help for exhaustively evaluating the consequences of our choices, discover possible new cases of deadlock involving contiguous or overlapping critical sections, and completely eliminate potential false positives situations from the newly introduced composite critical sections. As shown in Figure 3.7, we will rely on model checking approach for starting a sequence of iterations in which new problems in terms of deadlocks are found are resolved by creating and managing new sections in an incremental way.

3.5 A verifiable formal model of the system

The behavior of the abstract state machine describing the system can be rather easily formalized and verified in many ways and using different tools. We have chosen to follow a UML-like style of specification and exploit our in-house UMC framework.

UMC is an abstract, on-the-fly, state-event based, verification environment working on UML-like state machines [72]. Its development started at ISTI in

2003 and has been since then used in several research projects. So far UMC is not really an industrial scale project but more an (open source) experimental research framework. It is actively maintained and is publicly usable through its web interface (`https://fmt.isti.cnr.it/umc`).

In UMC a system is described as a set of communicating UML-like state machines. In our particular case the system is constituted by a unique state machine. The structure of a state machine in UMC is defined by a Class declaration which in general has the following structure:

```
class <name> is
    Signals:
      <list of asynchronous signals managed by the objects of the class>
    Operations:
      <list of synchronous call ops managed by the objects of the class>
    Vars:
      <list of local vars belonging to the state of the objects of the class>
    Behavior:
      <list of rules defining the state evolutions of the objects of the class>
end <name>
```

The Behavior part of a class definition describes the possible evolutions of the system. This part contains a list of transition rules which have the generic form:

```
<SourceState> --> <TargetState> {<EventTrigger>[<Guard> ] /<Actions> }
```

Each rule intuitively states that when the system is in the state SourceState, the specified EventTrigger is available, and all the Guards are satisfied, then all the Actions of the transition are executed and the system state passes from SourceState to TargetState (we refer to the UML2.0 [74] definition for a more rigorous definition of the run-to-completion step).

In UMC the actual structure of the system is defined by a set of active object instantiations. A full UMC model is defined by a sequence of Class and Objects declarations and by a final definition of a set of Abstraction rules. The overall behavior of a system is in fact formalized as an abstract doubly labelled transition system (L2TS), and the Abstraction rules allow to define what we want see as labels of the states and edges of the L2TS. The temporal

logic supported by UMC (which has the power of full μ-calculus but also supports the more high level operators of CTL/ACTL) uses this abstract L2TS as semantic model and allows to specify abstract properties in a way that is rather independent from the internal implementation details of the system [75].

It is outside the purpose of the chapter to give a comprehensive description of the UMC framework (we refer to the online documentation for more details). We believe instead that a detailed description of fragment of the overall system can give a rather precise idea of how the system is specified. To this purpose, we take into consideration just the top leftmost region of the railway yard as show by Figure 3.8, which is traversed only by the four trains of the green and red lines.

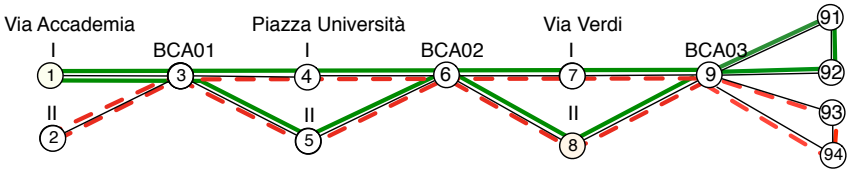


Figure 3.8: The top-left region of the full railway yard

Our UMC model is composed of a single class REGION1 and a single object SYS.

```
class  REGION1 is
    ...
end  REGION1
SYS:  REGION1  -- a single active object
Abstractions  {
     <observation rules>
    }
```

In our case the class REGION1 does not handle any external event, therefore the Signals and Operations parts are absent. The Vars part, in our case contains, for each train, the vector describing its mission, and a counter recording the current progress of the train (an index of the previous vector). E.g.

```
    G1M: int[]:= [1,3,4,6,7,9,92,91,9,8,6,5,3,1];   --mission of train Green1
    G1P: int  := 0;  --progress inside mission of train Green1, i.e. index in-
side G1M
```

Similar mission and progress data are defined for the other trains as G2M, G2P (train Green2), R1M, R1P (train Red1), R2M, R2P (train Red2).

As we have seen in the previous sections, in this area we have to handle six critical sections, called A, B, C, D, E, F. For the sake of simplicity, in this case we handle the linear A and B critical sections as if they were rings of size 2 (which allow at most one train inside them). We use six variables to record the limits of each section, and other six variables to record the current status of the various sections, properly initialized with the number of the trains initially inside them.

```
    MAXSA: int :=1;   -- section A: [1,3] (see Figures 3.3, 3.6a and 3.8)
    MAXSB: int :=1;   -- section B: [2,3]
    MAXSC: int :=3;   -- section C: [3,4,5,6]
    MAXSD: int :=3;   -- section D: [6,7,9,8]
    MAXSE: int :=3;   -- section E: [1,3,4,5,6]
    MAXSF: int :=3;   -- section F: [2,3,4,5,6]
    SA: int :=1;   SB: int :=1;   SC: int :=0;
    SD: int :=0;   SE: int :=1;   SF: int :=1;
```

For each train, the set of section updates to be performed at each step is recorded into another table which has the same size of the train mission. We show below the table `G1C` which describes the section operations to be performed by train Green1 during its progress:

```
    G1C: int[] :=  -- Section counters updates to be performed by train Green1
     --A,B,C,D,E,F
     [[1,0,0,0,1,0],     --1         [0,0,0,0,0,0],      --92-91
     [-1,0,1,0,0,1],     --1-3       [0,0,0,1,0,0],      --11-9
     [0,0,0,0,0,0],      --3-4       [0,0,0,0,0,0],      --9-8
     [0,0,-1,1,-1,-1],   --4-6       [0,0,1,-1,1,1],     --8-6
     [0,0,0,0,0,0],      --6-7       [0,0,0,0,0,-1],     --6-5
```

```
[0,0,0,-1,0,0],     --7-9       [1,0,-1,0,0,0],     --5-3
[0,0,0,0,0,0],      --9-92      [0,0,0,0,0,0]];     --3-1
```

The element `i` of the table records the increments or decrements that the train must apply to the various section counters to proceed from step `i` to step `i+1` of its mission. For example, in order to proceed, at step 1, from endpoint 1 to endpoint 3, train Green1 must apply the updates described in the element `[-1,0,1,0,0,1]`, i.e., decrement the counter of section `A`, and increment the counters for sections `C` and `F`.

In the Behavior part of our class definition we will have one transition rule for each train, which describes the conditions and the effects of the advancement of the train. In our case there is no external event which triggers the system transitions, therefore they will be controlled only by their guards.

In the case of train Green1, for example, we will have the rule:

```
01:  s1 -> s1
02:   { - [(G1P <13)  and  --  13 is the length of the mission for green1
03:     (G1M[G1P+1] /= R1M[R1P]) and                      ----
04:        (G1M[G1P+1] /= G2M[G2P]) and                       |
05:          (G1M[G1P+1] /= R2M[R2P]) and                     |
06:        (SA + G1C[G1P+1][0] <= MAXSA)  and                 |
07:          (SB + G1C[G1P+1][1] <= MAXSB)  and               | Guard
08:            (SC + G1C[G1P+1][2] <= MAXSC)  and             |
09:              (SD + G1C[G1P+1][3] <= MAXSD)  and           |
10:                (SE + G1C[G1P+1][4] <= MAXSE)  and         |
11:                 (SF + G1C[G1P+1][5] <= MAXSF)] /     ----
12:      SA := SA + G1C[G1P+1][0];      ----
13:      SB := SB + G1C[G1P+1][1];          |
14:      SC := SC + G1C[G1P+1][2];          |
15:      SD := SD + G1C[G1P+1][3];          | Actions
16:      SE := SE + G1C[G1P+1][4];          |
17:      SF := SF + G1C[G1P+1][5];          |
18:      G1P := G1P +1;                 ----
19:    }
```

The above rule states that, if train Green1 has not yet completed its mission (line 02), and the next endpoint for its mission is not already assigned to another train (lines 03–05), and for each critical section the update of its

associated counter does not exceed the stated limits (lines 06–11), then the train is allowed to proceed: the section counters are updated as requested by the step (lines 12–17) and the train progress is incremented of one step (line 18). Similarly, it is done for all the other four trains.

Finally, we have to define what we want to observe on the abstract L2TS associated to the system evolutions. Actually we are just interested to observe that a certain state is the final one, where all trains have completed all their steps, therefore returning to the point where they started from.

This can be done assigning a label, e.g. `ARRIVED` to all the system configuration in which the each train is in its final position.

```
Abstractions  {
State SYS.G1P=13  and
      SYS.G2P=13  and
      SYS.R1P=13  and
      SYS.R2P=13  ->  ARRIVED
}
```

The above abstraction rule specifies that the `ARRIVED` label should be assigned to a state when the progresses of the four trains reach the value 13 (the last index of all the train missions).

At this point the L2TS associated to our model will be a directed graph which will converge to a final state labelled ARRIVED in the case that no deadlock occurs in the system. This can be easily checked by verifying the CTL-like formula:

```
AF ARRIVED
```

The formula states that all paths (`A` in the formula) starting from the initial state of the system eventually will reach (`F`) a state labelled `HOME`. If this property does not hold we observe the generated counterexample and view all the details of the path which leads to the deadlocked state.

In our case the formula is true. The generated statespace has just 10073 configurations, and UMC explores all of them in a few seconds.

But are we sure that we have removed all the possible cases of false positives? One way to verify that is to allow a train to proceed even if its progress violates the constraints of the critical sections, but marking the reached state as DEAD. This is easily done in our model by removing the conditions on the section counters from the guards of the train transitions, and by adding in the Abstraction part the following observation rules:

```
State SYS.SA > MAXSA -> DEAD
State SYS.SB > MAXSB -> DEAD
State SYS.SC > MAXSC -> DEAD
State SYS.SD > MAXSD -> DEAD
State SYS.SE > MAXSE -> DEAD
State SYS.SF > MAXSF -> DEAD
```

In this way we can check the absence of false positives by verifying the formula:

```
not EF (DEAD and EF ARRIVED)
```

Which states that does not exists a path (E) which eventually reaches (F) a state that labelled DEAD, and from which exists (E) a continuation of the path which eventually reaches (F) a state in which all trains are in their destination.

Unfortunately the above formula false, and that allows discovering several other cases of false positives, (like the one shown in Figure 3.9) whose removal requires a more refined use of the counters (the final version of the code can be found at http://fmt.isti.cnr.it/umc/examples/traceit/).

If we want to check again the absence of deadlocks in this second kind of model we can now modelcheck the formula:

```
A[(EF ARRIVED) U (DEAD or ARRIVED)]
```

This is a typical branching time formula, which states two things. The first is that all paths will eventually reach a state labelled as DEAD or ARRIVED. The second is that for all intermediate states of these paths there is scheduling choice that allows driving all trains to destination (EF ARRIVED) .

In this case the size of the generated statespace is 10493 configurations and the evaluation time is less than two seconds.

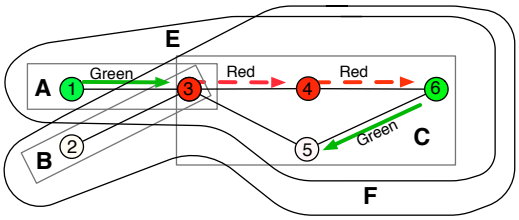


Figure 3.9: Another case of false positive for section E

3.6 Partitioning the Full Model

Sometimes the scheduling problem might be too complex to be handled by the model checker. In these cases, it is useful to split the overall layout into subregions to be analyzed separately. In particular, in the system used as our case study, we have four trains moving along the red-line and green-line service, and four other trains moving along the yellow-line and blue-line service. In we consider all the possible interleavings of eight trains each one performing about 20 steps, we get a system with about 20^8 configurations. Most model checkers (and UMC among them) may have difficulties in performing an exhaustive analysis over a system of this size, therefore it is useful to consider a possible splitting of the overall layout. In our case we have considered a partitioning of the system as shown in Figure 3.10. The analysis of region 1 has been performed following the approach outlined in the previous sections, and has led to the management of six critical sections.

The analysis of region 3 is similar to the previous one, and leads to the introduction of further four critical sections. The analysis of region 2 is more complex, being bigger and with 8 trains inside it. The analysis does not reveals any new cases of deadlocks or false positives, therefore the critical section remain the basic sections already discovered with our static analysis (shown in Figure 3.11). The statespace size of the model for region 2 is 6,820,504

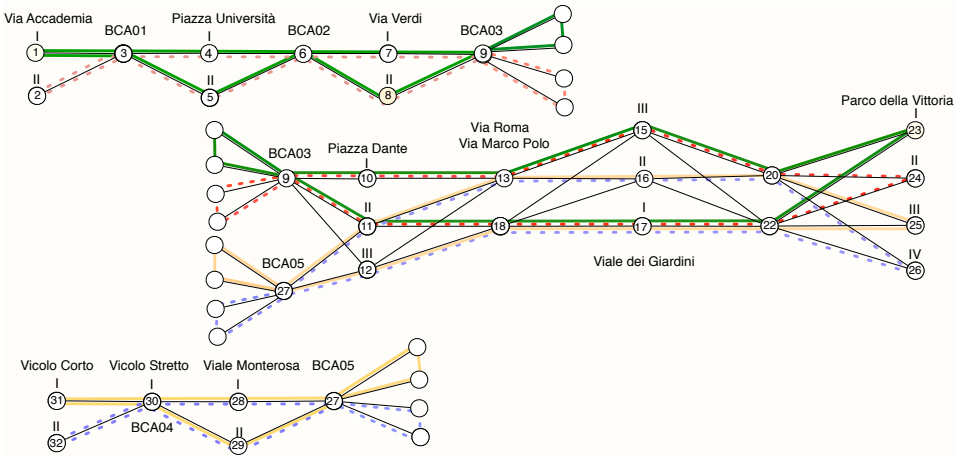


Figure 3.10: The three regions partitioning the full layout

configurations and its verification takes a few minutes.

In general, it is not true that the separate analysis of the single regions in which a layout is partitioned actually reveals all the possible deadlocks of the full system. For this being true it is necessary that the adopted partitioning does not cut (hiding it from the analysis) any critical section that overlaps two regions. Since we know from our static analysis where are positioned the basic critical sections for the layout, and we know that composite sections can only extend over contiguous/overapping basic sections, it is sufficient to partition the system in such a way that each region encloses completely a closed group of connected basic sections.

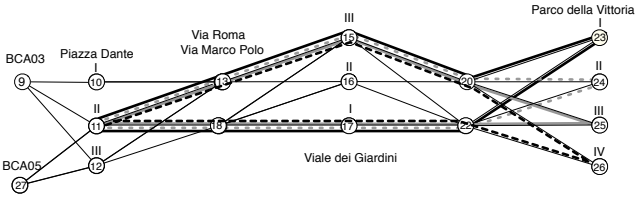


Figure 3.11: Four critical sections in region 2

Conclusions

This dissertation is based on the experience acquired inside the project namely "Train Control Enhancement via Information Technology" (TRACE-IT) funded by Tuscany Region. The project concerns the specification and development of a Communications-based Train Control (CBTC) platform, and sees the participation of the DINFO of the University of Florence, of the Formal Methods and Tools Laboratory of the "Institute of Information Science and Technologies" (ISTI), an institute of the "Italian National Research Council" (CNR) and E.C.M. s.p.a., an industrial partners.

In this dissertation, 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 dissertation. Another relevant aspect is the possibility to adapt the current approach to the development of Traffic Management System/European Train Control System (ERTMS/ETCS) sys-

tems[1]. This system intends to remove the technical barriers against the interoperability regarding the train control command system by creating a single Europe-wide standard. 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 highly 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.

To support the *feature model* definition, we presented an approach for commonality and variability mining from domain-specific natural language documents and we have presented two tools, namely CMT and FDE, which can ease domain analysis when a company wishes to enter a new market. They are used to semi-automate the process to define a *product family*. The two tools are both in a prototypical academic version, and several improvements are still needed to make them industrially applicable. Besides the look-and-feel improvements that are required, we plan to extend FDE with the introduction of minimum and maximum cardinalities in features and group of features. Moreover, we also plan to experiment the usage of the tools in real-world scenarios, to monitor how a user builds a feature model starting from NL documents. In our view, this user-based observation is fundamental to understand how to introduce feature-model synthesis approaches (as, e.g., in Davril *et al.* [76]) in a CMT/FDE-based tool-chain.

We also advocate the usage of the proposed tools – in particular CMT –

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

for mining common and optional features form NL requirements, and not only from informal product descriptions. In principle, requirements documents of similar products can be regarded as the brochures of different vendors, and processed according to the approach defined in this dissertation. In this case, the final output would be a feature model, that represents the product line associated to the requirements.

After, to improve completeness of the requirements we presented the novel concept of *backward functional completeness* of a requirements specification has been defined as the completeness of a specification with respect to the input documents of the requirements definition process. Metrics to measure such completeness have been provided, as well as a NLP-based tool named CAR to improve it. Further development of the principles of CAR are currently under analysis. We would like to give a *type* to the relations that are extracted from the input documents. For example, "ATS user interface" and "train schedule" are related in our input document, and their relation is of type "display". Furthermore, we would like to explore different approaches for choosing the terms to be suggested to the user of CAR. Such approaches should also take into account the structure of the input documents, the structure of the requirements specification itself, and the requirements previously written by the user. Other similarity metrics, such as the *cosine similarity*[77], are currently under analysis to evaluate the relations among the terms. After improving the principles of CAR, we plan to assess the tool with both academic and industrial case studies. In particular, we plan to consider systems of different domains, as well as different types of input documents, in order to identify possible refinements and domain-specific optimizations of the approach.

After studying the scenario of the CBTC system, we have decided to extend the use of the Formal Methods to manage *degraded modes operation*, in the development of an ATS system. This choice stems from the need to manage situations that would have great impact on the operation of the railway system to avoid the block of the railway network for many hours. The development of solutions to the problem of deadlock avoidance in train scheduling is a complex and still open task [78]. Many studies have been carried out on the subject

since the early '80s, but most of them are related to normal railway traffic, and not to the special case of driverless metropolitan systems. Automatic metro systems indeed may express some original properties, e.g., the difficulty of changing the station platform on which a train should stop, or the fact that all trains keep moving continuously, which makes the problem rather different from the classical railway case. There are many directions in which this work is going to proceed. For example, we want to see if the model checking / model refinement cycles for the detection and management of critical sections could be in some way fully automatized removing the human intervention for the generation of the final validated ATS configuration data. A further interesting evolution would be the generation and validation of the critical sections data directly from the inside of the ATS. This would allow to automatically handle at run time also the dynamic change of the itinerary of the trains. The current metro-line oriented approach could be further generalized to a wider railway oriented setting by taking into consideration the train and platform lengths, or the possibility of specifying connections and overtakings among trains. At a first look the handling of these aspects should require only minor updates of our current approach.

Therefore *agile* approaches are useful to start developing, but formal methods are useful afterwards. It is not indispensable to analyse all parts of the system with formal methods, but only the most critical sub-parts. After identifying the sub-components that need formal methods, such methods can be applied.

Bibliography

[1] R. D. Pascoe and T. N. Eichorn, "What is Communication-Based Train Control?," *IEEE Vehicular Technology Magazine*, 2009.

[2] 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.

[3] P. C. Clements and L. Northrop, *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley Longman, Inc., 2001.

[4] A. Fantechi and S. Gnesi, "Formal modeling for product families engineering," in *Proc. of SPLC*, pp. 193–202, 2008.

[5] CENELEC, "EN 50128, Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems," 2011.

[6] CENELEC, "EN 50126, Railway applications - the specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - part 1: Generic RAMS process," 2012.

[7] S. Black, P. Boca, J. Bowen, J. Gorman, and M. Hinchey, "Formal Versus Agile: Survival of the Fittest," *Computer*, vol. 42, pp. 37–45, Sept 2009.

[8] P. G. Larsen, J. S. Fitzgerald, and S. Wolff, "Are Formal Methods Ready for Agility? A Reality Check," in *FM+AM 2010 - Second International Workshop on Formal Methods and Agile Methods, 17 September 2010, Pisa (Italy)*, pp. 13–25, 2010.

[9] B. Fitzgerald, K.-J. Stol, R. O. Sullivan, and D. O. Brien, "Scaling Agile Methods to Regulated Environments: An Industry Case Study," in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, (Piscataway, NJ, USA), pp. 863–872, IEEE Press, 2013.

[10] S. Wolff, "Scrum goes formal: Agile methods for safety-critical systems," in *Software Engineering: Rigorous and Agile Approaches (FormSERA), 2012 Formal Methods in*, pp. 23–29, June 2012.

[11] X. Ge, R. Paige, and J. McDermid, "An iterative approach for development of safety-critical software and safety arguments," in *Agile Conference (AGILE), 2010*, pp. 35–43, Aug 2010.

[12] H. Jonsson, S. Larsson, and S. Punnekkat, "Agile Practices in Regulated Railway Software Development," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, pp. 355–360, Nov 2012.

[13] "IEC 62290-1: Railway applications: Urban guided transport management and command/control systems. Part 1: System principles and fundamental concepts," 2007.

[14] "IEC 62290-2: Railway applications: Urban guided transport management and command/control systems. Part 2: Functional requirements specification," 2011.

[15] CENELEC, "EN 50129, Railway applications - Communications, signalling and processing systems - Safety related electronic systems for signalling," 2003.

[16] J. S. Stover, "CITYFLO 650 System Overview." http://goo.gl/e26SZ, 2006.

[17] Signalling Solutions Limited, "URBALIS Communication Based Train Control (CBTC) Delivery Performance and Flexibility." http://goo.gl/G3hEe, 2009.

[18] Thales Transportation, "Seltrac Brochure." http://goo.gl/OjhvK, 2009.

[19] Invensys Rail, "SIRIUS Brochure." http://goo.gl/YFUiL, 2009.

[20] Ansaldo STS, "CBTC Brochure." http://goo.gl/3Kmb0, 2011.

[21] Siemens Transportation Systems, "Trainguard MT CBTC." http://goo.gl/Xi0h0, 2006. The Moving Block Communications Based Train Control Solution.

[22] GE Transportation, "Tempo CBTC Solution." http://goo.gl/KshrR, 2012.

[23] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," tech. rep., Carnegie-Mellon University Software Engineering Institute, 1990.

[24] D. S. Batory, "Feature models, grammars, and propositional formulas," in *Proc. of SPLC*, pp. 7–20, 2005.

[25] K. Czarnecki and U. Eisenecker, *Generative programming: methods, tools, and applications*. New York, NY, USA: ACM Press/Addison-Wesley, 2000.

[26] A. Sutcliffe, "Scenario-based requirements engineering," in *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, RE '03, (Washington, DC, USA), pp. 320–329, IEEE Computer Society, 2003.

[27] H. Gomaa, "The impact of rapid prototyping on specifying user requirements," *SIGSOFT Softw. Eng. Notes*, vol. 8, pp. 17–27, Apr. 1983.

[28] 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*, pp. 97–105, IEEE, 2001.

[29] E. Kuun, "Open Standards for CBTC and CBTC Radio Based Communications," in *APTA Rail Rail Transit Conference Proceedings*, 2004.

[30] A. Ferrari, G. O. Spagnolo, and F. dell'Orletta, "Mining commonalities and variabilities from natural language documents," in *SPLC* (T. Kishi, S. Jarzabek, and S. Gnesi, eds.), pp. 116–120, ACM, 2013.

[31] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[32] F. Roos-Frantz, *Automated Analysis of Software Product Lines with Orthogonal Variability Models: Extending the FaMa Ecosystem*. PhD thesis, University of Seville, 2012.

[33] M. Mendonca, M. Branco, and D. Cowan, "Splot: software product lines online tools," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pp. 761–762, ACM, 2009.

[34] A. Ferrari, A. Fantechi, G. Magnani, D. Grasso, and M. Tempestini, "The metrô rio case study," *Sci. Comput. Program.*, vol. 78, no. 7, pp. 828–842, 2013.

[35] A. Ferrari, A. Fantechi, S. Gnesi, and G. Magnani, "Model-based development and formal methods in the railway industry," *IEEE Software*, vol. 30, no. 3, pp. 28–34, 2013.

[36] R. Schwitter, "English as a formal specification language," in *DEXA Workshops*, pp. 228–232, IEEE Computer Society, 2002.

[37] C. Grover, A. Holt, E. Klein, and M. Moens, "Designing a controlled language for interactive model checking," in *Proceedings of the Third International Workshop on Controlled Language Applications*, pp. 29–30, 2000.

[38] 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*, pp. 339–349, 2005.

[39] A. Bucchiarone, S. Gnesi, A. Fantechi, and G. Trentanni, "An experience in using a tool for evaluating a large set of natural language requirements," in *SAC* (S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C.-C. Hung, eds.), pp. 281–286, ACM, 2010.

[40] A. Ferrari and S. Gnesi, "Using collective intelligence to detect pragmatic ambiguities," in *Requirements Engineering Conference (RE), 2012 20th IEEE International*, pp. 191–200, IEEE, 2012.

[41] A. Fantechi, S. Gnesi, A. Lapadula, F. Mazzanti, R. Pugliese, and F. Tiezzi, "A logical verification methodology for service-oriented computing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 21, no. 3, p. 16, 2012.

[42] M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti, "A state/event-based model-checking approach for the analysis of abstract system properties," *Science of Computer Programming*, vol. 76, no. 2, pp. 119–135, 2011.

[43] R. Nelken and N. Francez, "Automatic translation of natural language system specifications into temporal logic," in *Computer Aided Verification*, pp. 360–371, Springer, 1996.

[44] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini, "Assisting requirement formalization by means of natural language translation," *Formal Methods in System Design*, vol. 4, no. 3, pp. 243–263, 1994.

[45] K. Schwaber, *Agile project management with Scrum*. Microsoft Press, 2004.

[46] G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel, "Product Line Analysis: A Practical Introduction," Tech. Rep. CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, 2001.

[47] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," in *Proc. of ICSE '97*, (New York, NY, USA), pp. 161–171, ACM, 1997.

[48] A. Ferrari, S. Gnesi, and G. Tolomei, "Using clustering to improve the structure of natural language requirements documents," in *Requirements Engineering: Foundation for Software Quality* (J. Doerr and A. L. Opdahl, eds.), vol. 7830 of *Lecture Notes in Computer Science*, pp. 34–49, Springer Berlin Heidelberg, 2013.

[49] G. G. Chowdhury, "Natural language processing," *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[50] F. Bonin, F. Dell'Orletta, S. Montemagni, and G. Venturi, "A contrastive approach to multi-word extraction from domain-specific corpora," in *Proc. of LREC'10*, pp. 19–21, 2010.

[51] M. Mendonca, M. Branco, and D. Cowan, "S.p.l.o.t.: Software product lines online tools," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pp. 761–762, ACM, 2009.

[52] F. Dell'Orletta, G. Venturi, A. Cimino, and S. Montemagni, "T2kˆ2: a system for automatically extracting and organizing knowledge from texts," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), Reykjavik, Iceland, May 26-31, 2014.*, pp. 2062–2070, 2014.

[53] F. Dell'Orletta, "Ensemble system for part-of-speech tagging," in *Proc. of Evalita'09, Evaluation of NLP and Speech Tools for Italian*, 2009.

[54] S. Tan, "Neighbor-weighted k-nearest neighbor for unbalanced text corpus," *Expert Systems with Applications*, vol. 28, no. 4, pp. 667–671, 2005.

[55] P. Rayson, R. Garside, and P. Sawyer, "Recovering legacy requirements," in *Proc. of REFSQ'99)*, pp. 49–54, 1999.

[56] K. Pohl, G. Böckle, and F. Van Der Linden, *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.

[57] L. Goldin and D. M. Berry, "Abstfinder, a prototype natural language text abstraction finder for use in requirements elicitation," *Autom. Softw. Eng.*, vol. 4, no. 4, pp. 375–412, 1997.

[58] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with CIRCE," *Autom. Softw. Eng.*, vol. 13, no. 1, pp. 107–167, 2006.

[59] A. M. Hickey and A. M. Davis, "Requirements elicitation and elicitation technique selection: model for two knowledge-intensive software development processes," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, pp. 10–pp, IEEE, 2003.

[60] S. Lauesen, *Software Requirements: Styles and Techniques*. Addison-Wesley, 2002.

[61] B. Boehm, "Verifying and validating software requirements and design specifications," *Software, IEEE*, vol. 1, no. 1, pp. 75–88, 1984.

[62] D. Zowghi and V. Gervasi, "The Three Cs of Requirements: Consistency, Completeness, and Correctness," in *Proc. of REFSQ'02*, pp. 155–164, 2002.

[63] O. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *Software, IEEE*, vol. 11, no. 2, pp. 42–49, 1994.

[64] S. España, N. Condori-Fernandez, A. Gonzalez, and O. Pastor, "Evaluating the completeness and granularity of functional requirements specifications: A controlled experiment," in *Proc. of RE '09*, pp. 161–170, 2009.

[65] B. Yadav, Surya, R. Bravoco, Ralph, T. Chatfield, Akemi, and T. M. Rajkumar, "Comparison of analysis techniques for information requirement determination," *Commun. ACM*, vol. 31, pp. 1090–1097, Sept. 1988.

[66] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos, "Identifying and measuring quality in a software requirements specification," in *Proc. of SMS'93*, pp. 141–152, 1993.

[67] R. J. Costello and D.-B. Liu, "Metrics for requirements engineering," *J. Syst. Softw.*, vol. 29, pp. 39–63, Apr. 1995.

[68] I. Menzel, M. Mueller, A. Gross, and J. Dörr, "An experimental comparison regarding the completeness of functional requirements specifications," in *Proc. of RE '10*, pp. 15–24, 2010.

[69] H. Kaiya and M. Saeki, "Ontology based requirements analysis: lightweight semantic processing approach," in *Quality Software, 2005. (QSIC 2005). Fifth International Conference on*, pp. 223–230, 2005.

[70] T. Dunning, "Accurate methods for the statistics of surprise and coincidence," *Comput. Linguist.*, vol. 19, pp. 61–74, Mar. 1993.

[71] A. Fantechi, W. Fokkink, and A. Morzenti, "Some Trends in Formal Methods Applications to Railway Signaling," in *Formal Methods for Industrial Critical Systems: A Survey of Applications*, pp. 61–84, John Wiley & Sons, 2013.

[72] S. Gnesi and F. Mazzanti, "An abstract, on the fly framework for the verification of service-oriented systems," in *Rigorous Software Engineering for Service-Oriented Systems*, vol. 6582 of *LNCS*, pp. 390–407, Springer, 2011.

[73] F. Mazzanti, G. O. Spagnolo, and A. Ferrari, "Designing a deadlock-free train scheduler: A model checking approach," in *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*, pp. 264–269, 2014.

[74] OMG, "Object Management Group, UML Superstructure Specification http://www.omg.org/spec/UML/2.4.1," 2006.

[75] F. ter Beek, M.H.and Mazzanti and S. Gnesi, "CMC-UMC: A Framework for the Verification of abstract Service-Oriented Properties," in *Proceedings of the 2009 ACM symposium on Applied Computing*, pp. 2111–2117, ACM, 2009.

[76] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pp. 290–300, ACM, 2013.

[77] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*, vol. 1. Cambridge university press, 2008.

[78] J. Törnquist, "Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms," in *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, p. 659, 2006.