SOSYM manuscript No.
(will be inserted by the editor)

Formalizing requirements with object models and temporal constraints

Alessandro Cimatti, Marco Roveri, Angelo Susi, Stefano Tonetta [⋆]

e-mail: {cimatti,roveri,susi,tonettas}@fbk.eu

Fondazione Bruno Kessler - IRST

Via Sommarive 18, 38050 Povo (TN) Italy

Received: date / Revised version: date

Abstract Flaws in requirements often have a negative impact on the subsequent development phases. In this paper, we present a novel approach for the formal representation and validation of requirements, which we used in an industrial project. The formalism allows us to represent and reason about object models and their temporal evolution. The key ingredients are class diagrams to represent classes of objects, their relationships and their attributes, fragments of first order logic to constrain the possible configurations of such objects, and temporal logic operators to deal with the dynamic evolution of the configurations. The approach to formal validation allows to check whether the requirements are consistent, if they are compatible with some scenarios, and if they guarantee some implicit properties.

[⋆] Supported by the Provincia Autonoma di Trento (project ANACONDA).

The validation procedure is based on satisfiability checking, which is carried out by means of finite instantiation and model checking techniques.

Key words Formal requirement engineering; temporal logic; railway domain; ETCS

1 Introduction

Often problems encountered in late development phases can be traced back to flaws in the requirements. For this reason, the development of techniques and tools for the analysis of requirements is an important objective, in particular for safety-critical applications and/or software. Formal methods can provide substantial support, by providing mathematically precise representation languages and inference mechanisms.

In the context of hardware design and verification, formal languages to specify the requirements have been standardized [1, 2] and the validation techniques have been engineered. The formal language used is a combination of temporal logic with regular expressions, while the validation is performed with a series of checks that aim at assuring that the formal specification correspond to what the designer intended to write. The validation checks are accomplished with algorithms to solve the satisfiability of temporal formulas.

In the context of safety-critical applications, the choice of the language used to formalize the requirements is still an open issue, requiring a delicate balance between expressiveness, decidability, and complexity of inference. The difficulty in finding a suitable trade-off lies in the fact that the requirements for many real-world applications involve several dimensions. On the one side, the objects having an active role in the target application may have complex structure and mutual relationships, whose modeling may require the use of rich data types. On the other side, static constraints over their attributes must be complemented with constraints on their temporal evolution.

In this paper, we address the problem of validating requirements in complex domains, by presenting the core of a framework used for the formalization and validation of railways requirements in an industrial project.

We define the syntax and semantics of a rich specification language, which combines the elements of several formalisms in order to enable a natural modeling. The objects and their attributes are characterized using class diagrams, which induce the signature of our logic. The logic includes first-order formulas with quantifiers over the objects of a class. Temporal constraints are expressed by means of temporal operators, resulting in a fragment of first order temporal logic [28,29] (the fragment depends on the particular adopted first-order theory). The language extends the one presented in [12] trading-in the automation of the analysis. In fact, we allow quantifiers to range over the objects of a certain class and to be interleaved with temporal operators (while in [12] quantifications were only allowed to range over the elements of attributes and they were restricted to occur within atomic formulas).

The validation of the requirements is based on an approach that combines finite instantiation of objects and a reduction to model checking problems. Each validation check defines a finite instantiation of the objects and a set of constraints over some classes. We exploit the techniques described in [12] to avoid limiting the number of objects for non-constrained classes. The satisfiability procedure is based on several reduction steps: first, we obtain a first-order temporal formula φ which is equi-satisfiable to the original specification. and free of object terms and free of quantifiers; second, we create a fair transition system whose accepted language is non-empty iff φ is satisfiable; finally, we check for the language accepted by the fair transition system being non-empty. The resulting problem is in turn addressed with automatic model checking techniques. Emerging symbolic model checking techniques, based on the use of SMT solvers and abstraction [26,7], can be used to efficiently check for the non-emptiness of the language of the fair transition system. The described approach has been implemented within an extended version of the NuSMV model checker, which is able to verify infinite-state systems.

The language has been used by railway experts within the EuRailCheck project funded by the European Railway Agency, with the purpose of formalizing and validating a significant subset of the ETCS specification. We consider as case study some paradigmatic requirements and we prove that the specification is satisfiable, that few desired scenarios are allowed, and that a property holds.

The paper is structured as follows. In Section 2 we present a motivating application domain. In Section 3 we formalize the notion of class diagrams, and in Section 4 we introduce the formalism used to specify constraints. In Section 5 we present our approach to requirements validation, and in Section 6 we discuss the reduction procedure for checking satisfiability. In Section 7 we present the implementation of the method and the experimental analysis aiming at showing the applicability of the proposed approach on a subset of a real case study. In Section 8 we compare our approach with the state of the art, and in Section 9 we draw conclusions and discuss directions for future work.

2 Motivating Application Domain

2.1 Formalizing and validating the ETCS specification

The European Train Control System (ETCS) [19] is a specification that provides a standard for train control systems to guarantee the interoperability with trackside system across different European countries. The ETCS specification is a set of requirements related to the automatic supervision of the location and the speed of the train performed by the on-board system.

ETCS is already installed on important railway lines in different European countries. It is based on the implementation on board of a set of safety critical functions of speed and distance supervision and of information to the driver. Such functions rely on data transmitted by trackside installations through two communication channels: physical devices that lie on the track, called balises, and radio communication with control centers, called Radio Block Centers (RBCs).

In 2007, the European Railway Agency (ERA) issued a call for tender for the development of a methodology complemented by a set of support tools, for the formalization and validation of the ETCS specifications.

During the EuRailCheck project [21], that originated from the successful response to the call for tender, we developed a requirements validation framework [11].

The aim of this framework consists in analyzing the ETCS specifications, in order to eliminate various kinds of flaws. First, we exploit the framework identifies and eliminate ambiguities during the formalization, so that the requirements can be uniquely interpreted by the system designers. This is particularly important in the case of ETCS, since the requirements are to be applied in different cultural contexts such as the different national railway manufacturers. Second, via the framework it should be possible to detect inconsistencies in the specifications, which would invalidate any system verification.

In order to do this, we rely on formal specification, which guarantees the improvement of the clarity of the requirements, and on formal verification, in order to automate the analysis and the search for possible flaws. We remark that the project poses requirements that are significantly different from "more traditional" applications of formal methods. Traditionally, in formal verification, the entity under analysis is a design, which is to be verified against a set of requirements, assumed to be correct. Here, the purpose of the activity is the analysis of the ETCS requirements, and thus the use of formal methods, traditionally oriented to design verification, has to be re-thought before being applicable.

The main issue in achieving these goals is that ETCS is a huge complex system, with many functional modules, which need to be clearly separated between on-board and trackside functionalities, and is contributed by a vast set of people. Many ambiguities come from the use of natural language, from corrections applied by different people, with different cultures, and of different mother tongues.

For these reasons, the ETCS specification poses demanding requisites to the formal methods adopted for its validation. First, the formalism should be able to capture the meaning of the requirements. Second, it should be as simple as possible to be used by non-experts in formal methods: the requirements are usually ambiguous English sentences that only an expert in the domain can formalize and validate.

A property-based approach to requirements validation relies on the availability of an expressive temporal logic, so that each informal statement has a formal counterpart with similar structure. This is in contrast with the model-based approach that formalizes the requirements with a high-level design and requires more efforts in the formalization and its traceability.

A natural choice of the logic is the Linear-time Temporal Logic (LTL) [31] extended with Regular Expressions (RELTL) [6] and first-order predicates [28]. The temporal formulas often resemble their informal counterparts because many natural language words used for temporally connecting events can be precisely mapped to operators in the temporal logic. First-order predicates have already been proven adequate to describe relationships among objects (see, e.g., [25]). Finally, regular expressions have been proven valuable in specification languages such as PSL [1], for describing sequences of events.

We show how complex statements find a natural formalization in the proposed language through a running example. This is of paramount importance when experts in the application domain have the task of disambiguating and formalizing the requirements.

2.2 Running example

Let us consider parts of the ETCS specification describing the interoperability between trains and trackside. In particular, we refer to the System Requirements Specification (SRS), subset 26 of the ETCS [20].

Focusing on a set of requirements describing the interaction between trains and trackside, some objects, called *balises*, are located on the track and are able to send messages to the trains in order to communicate/refine the information about the train position. In particular, the balises are usually positioned sequentially on the track in groups of different size called *balise group*s. Each balise stores its *internal number* in the group (in general from 1 to 8), the *identity of the balise group* and the *number of the balises inside the group* the balise belongs to. Each balise group maintains also a *linking information* describing the position of the next balise group on the track. Each balise sends messages to the train, the *on-board*, passing over it. The messages, called *telegrams*, inform the train about its position and orientation on the track, and about the next balise group on the track (via the *linking information*). In case the train cannot understand its orientation (simply using the information from the balise groups, or the train does not have the *linking information* available), the train has to memorize the information of several balise groups found during the trip (and in particular of the *Last Relevant Balise Group*), send this information to the *Radio Block Center* (RBC) device, which is a trackside system communicating via radio with the on-board, and wait for an answer by the RBC. Figure 1 illustrates this scenario.

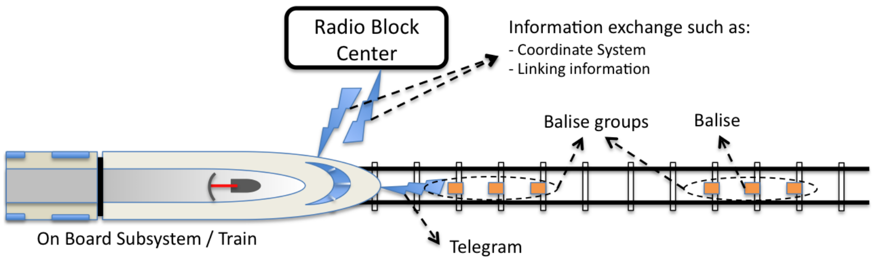

Fig. 1 The scenario of the train running on a track with the signalling devices and information exchanges.

The requirements in Table 1 are excerpts of the ETCS specification describing part of the scenario envisaged so far, that have been formalized during the EuRailCheck project by domain experts. We use these requirements as running example to explain the techniques presented in this paper.

3 Class Diagrams

Our work considers as fundamental modeling concepts classes of objects. We use the notation of UML2 class diagrams (we refer the reader to the OMG UML2 metamodel specification documents [35] for a detailed description of these concepts). The choice of the UML2 visual modeling language, and in particular of the classes and class diagrams offered by this language, has been motivated by the need to easily represent the relevant entities of the domain under analysis and their relationships. UML2 is a well established language in Software Engineering; the fact that it is a visual language is an important characteristic in its exploitation in the software engineering process, in particular for the requirements and design phases.

Req. id.	SRS Code	Requirements
R#1	**3.4.1.1**	A balise group shall consist of between one and eight balises.
R#2	**3.4.1.2**	In every balise shall at least be stored:
R#2.a	*3.4.1.2.a*	the internal number (from 1 to 8) of the balise
R#2.b	*3.4.1.2.b*	the number of balises inside the group
R#2.c	*3.4.1.2.c*	the balise group identity.
...	...	...
R#3	**3.4.1.3**	The internal number of the balise describes the relative position of the balise in the group.
R#4	**3.4.2.3.3.1**	If [...] no linking information is available on-board, the RBC shall be requested to assign a co-ordinate system as follows
...	...	...
R#4.d	*3.4.2.3.3.1.d*	The Last Relevant Balise Groups (LRBG) reported to the Radio Block Center (RBC) shall be memorised by the on-board equipment, [...], until the RBC has assigned a co-ordinate system.
...	...	...
R#5	**3.16.2.1.1**	The information that is sent from a balise is called a balise telegram.

Table 1 An excerpt of the ETCS specification formalized during the EuRailCheck project.

Definition 1 *A class diagram D consists of:*

Primitive types *A finite set of primitive types $PT = \{\tau_1, \ldots, \tau_h\}$.*

Classes *A finite set of classes $C = \{c_1, \ldots, c_n\}$.*

Attributes *For each class $c \in C$, a finite set of attributes $c.A = \{c.a_1, \dots, c.a_m\}$.*

Every attribute $c.a$ has

– *a type $c.a.\mathit{Type} \in C \cup \mathit{PT}$;*

– *a multiplicity $c.a.\mathit{Mult}$ that defines an interval of natural numbers; we denote with $m..n$, $0 \leq m \leq n$, the interval $\{i | m \leq i \leq n\}$ and with $m..*$, $m \geq 0$, the interval $\{i | m \leq i\}$; we write $\min(c.a.\mathit{Mult})$ for the lower bound (m), and $\max(c.a.\mathit{Mult})$ for the upper bound (n or $*$).*

To simplify the presentation, we limit the description to a core subset of the UML2 class diagrams which only encompasses the concepts of classes and class attributes. We also assume that the attributes are *sequences* of elements. We remark that our techniques capture UML2 associations, aggregations, inheritance, and can be extended to handle the other UML2 collection types such as *sets* and *bags*. Each attribute has a type and a multiplicity $m..n$, with $0 \leq m \leq n$, where m and n indicate the minimum and maximum number of elements, respectively. When the multiplicity is $m..*$, the number of elements in unbounded. The case of multiplicity equal to $1..1$ is treated in a special manner, since the collection has exactly one element.

3.1 Running example

The set of UML2 classes represented in Figure 2 defines the ontology of our case study, as specified in the set of requirements we introduced in Table 1. Each class contains a list of attributes with the type and the multiplicity. If not specified, the multiplicity is $1 \dots 1$.

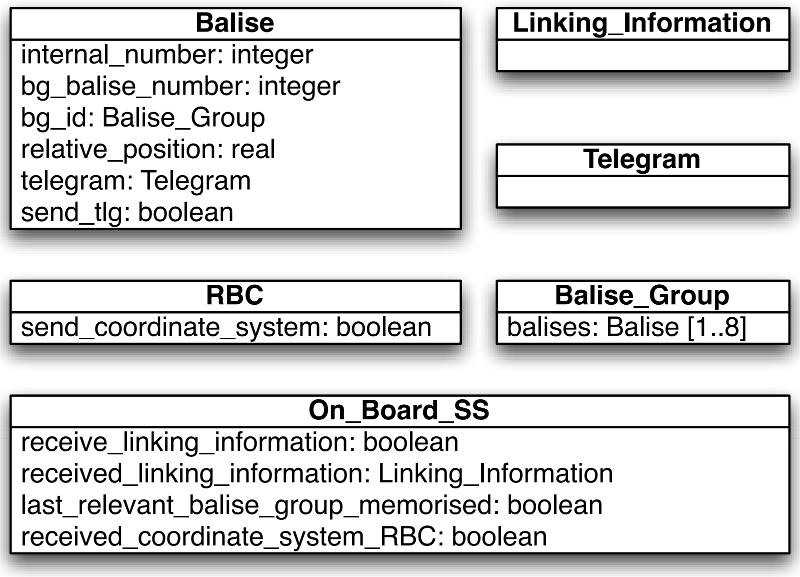
Balise
internal_number: integer
bg_balise_number: integer
bg_id: Balise_Group
relative_position: real
telegram: Telegram
send_tlg: boolean
Linking_Information
Telegram
RBC
send_coordinate_system: boolean
Balise_Group
balises: Balise [1..8]
On_Board_SS
receive_linking_information: boolean
received_linking_information: Linking_Information
last_relevant_balise_group_memorised: boolean
received_coordinate_system_RBC: boolean

Fig. 2 The classes representing the basic concepts described by the requirements of Table 1.

In particular:

– To represent the requirement R#1, we introduced two classes: the `Balise` and the `Balise_Group`. Moreover, we also introduced an attribute of the class `Balise_Group`, called `balises` that is an array of type `Balise` containing from 1 to 8 `Balises`.
– To formalize the three statements of the requirement R#2 we introduced three attributes in the class `Balise`: `internal_number`, for R#2.a, `bg_balise_number`, for R#2.b and `bg_id` to formalize R#2.c.
– To formalize R#3 we introduced the attribute `relative_position` in class `Balise`.

– To formalize R#4 we introduced: the class `On_Board_SS`, to formalize the on-board sub-system; the class `RBC` to represent the Radio Block Center; the class `Linking_Information` to represent the linking information; the attributes `receive_linking_information` and `received_linking_information` in the class `On_Board_SS`, to translate the reception of the linking information; the attribute `received_coordinate_system` of the class `On_Board_SS`. The paragraph R#4.d of the requirement introduces the last relevant balise group that is stored on-board, so it becomes the attribute `last_relevant_balise_group_memorised` of the class `On_Board_SS`.
– the requirement R#5 introduces the concept of balise telegram, which is sent from the balise to the on-board; this is translated into a new class `Telegram` and two attributes `telegram` and `send_tlg` of the class `Balise`.

4 Temporal Constraints

4.1 Syntax

We consider a fragment of the First-Order Temporal Logic of Manna and Pnueli ([29]), and we combine temporal operators with regular expressions in order to get ω-regular expressiveness [27].

We assume that some requirements have already been formalized with some classes and attributes. The temporal constraints are used to annotate the class diagram in order to formalize further requirements. Thus, the syntax of the language depends on a given class diagram D with a set C of classes and a set *PT* of primitive types. We assume to have an underlying first-order signature Σ. The symbols in Σ can express relations over the domains of the primitive types. For example, Σ can contain constants and functions over Booleans, Reals, Bounded Integers, and Integers. We denote with $\textsc{Coll}_\tau$ for $\tau \in C \cup \mathit{PT}$ the type representing the collection of elements of type τ. Finally, we assume to be given a set of variables $\textsc{var}_\tau$ for each type $\tau \in C \cup \mathit{PT}$.

We define the temporal language TL_D as follows. Let $\textsc{var}$, $\textsc{const}$, $\textsc{fun}$, $\textsc{rel}$, $\textsc{type}$, and $\textsc{attr}$ be respectively the set of variables, Σ constant symbols, Σ function symbols, Σ relational symbols, type symbols, and attributes symbols.

Definition 2 **(TL_D static terms)** *The set* $\textsc{term}$ *of terms is defined as follows:*

$$\textsc{term} := \textsc{var} \mid \textsc{const} \mid \textsc{fun}(\textsc{term}, \ldots, \textsc{term}) \mid \textsc{term}.\textsc{attr} \mid \textsc{term}.\textsc{attr}.\mathit{size} \mid \textsc{term}.\textsc{attr}[\textsc{term}]$$

For each term $f(t_1, \ldots, t_n) \in \textsc{term}$, if the type of f is $\tau_1 \rightarrow \ldots \rightarrow \tau_n \rightarrow \tau$, then $t_1, \ldots, t_n$ must be terms of type $\tau_1, \ldots, \tau_n$. For each term $t.a$, if the type of t is $c \in C$, then it is required that $a \in c.A$. For each term $t.a.size$ or $t.a[i]$, if the type of t is $c \in C$, then it is required $a \in c.A$, $a.Mult \neq 1..1$ and i to be a term of type $a.Mult$.

Respecting the UML2 standard [35], if $a.\mathit{Type} = \tau$ and $a.\mathit{Mult} = 1..1$, then $t.a$ if a term of type τ, otherwise $t.a$ is a term of type $\textsc{Coll}_\tau$. For example, if v is a variable of class c and a is an Integer attribute in $c.A$, then if $a.\mathit{Mult} = 1..1$ then $v.a$ is an Integer, otherwise $v.a$ is a collection of Integers.

Definition 3 (TL_D **temporal terms)** *The set* TTERM *of temporal terms is defined as follows:*

$$\text{TTERM} := \text{TERM} \mid \textit{next}(\text{TERM})$$

Atomic formulas combine terms with relational symbols and include Boolean operations and quantifiers. As in PSL, they are used as atoms of regular expressions and temporal formulas allowing a compact and intuitive specification (w.r.t. languages where atomic formulas cannot contain Boolean operations and quantifiers).

Definition 4 (TL_D **atomic formulas)** *The set* ATOM *of atomic formulas is defined as follows:*

$$\begin{aligned}\text{ATOM} := {} & \text{REL}(\text{TERM}, \ldots, \text{TERM}) \mid \text{TERM} = \text{TERM} \mid \text{TERM} \in \text{TERM} \mid \\ & \neg\text{ATOM} \mid \text{ATOM} \wedge \text{ATOM} \mid \\ & \forall \text{VAR} \in 1..\text{TERM}.\text{ATTR}.\textit{size}\ (\text{ATOM}) \mid \forall \text{VAR} : \text{TYPE}\ (\text{ATOM})\end{aligned}$$

For each atomic formula $r(t_1, \ldots, t_n) \in$ TERM, if the type of r is $\tau_1 \times \ldots \times \tau_n$, then $t_1, \ldots, t_n$ must be terms of type $\tau_1, \ldots, \tau_n$. For each atomic formula $t_1 = t_2$, t_1 and t_2 must be of the same type. For each atomic formula $t_1 \in t_2$, if t_1 is a term of type τ, t_2 must be of type COLL_τ. For each atomic formula $\forall i \in 1..t.a.\textit{size}\ (\varphi)$, it is required $a \in c.A$, $a.\textit{Mult} \neq 1..1$ and i to be a variable of type $a.\textit{Mult}$.

Definition 5 (TL_D **regular expressions)** *The set* SERE *of regular expressions is defined as follows:*

$$\begin{aligned}\text{SERE} := {} & \text{ATOM} \mid \epsilon \mid \text{SERE}* \mid \text{SERE}; \text{SERE} \mid \text{SERE} : \text{SERE} \mid \\ & \text{SERE} \vee \text{SERE} \mid \text{SERE} \wedge \text{SERE}\end{aligned}$$

Definition 6 **(TL_D formulas)** *The set* FORM *of formulas is defined as follows:*

$$\begin{aligned} \text{FORM} := {} & \text{ATOM} \mid \neg\text{FORM} \mid \text{FORM} \wedge \text{FORM} \mid \\ & \mathbf{X}\ \text{FORM} \mid \text{FORM}\ \mathbf{U}\ \text{FORM} \mid \text{SERE}\ \mid\!\!\rightarrow \text{FORM} \mid \\ & \forall\text{VAR} \in 1..\text{TERM.ATTR}.\mathit{size}\ (\text{FORM}) \mid \forall\text{VAR} : \text{TYPE}\ (\text{FORM}) \end{aligned}$$

For each formula $\forall i \in 1..t.a.size\ (\varphi)$, it is required $a \in c.A$, $a.Mult \neq 1..1$ and i to be a variable of type $a.Mult$.

We use the following standard abbreviations:

– *true* $\equiv v \neq v$ for some variable v;

– *false* $\equiv \neg$*true*;

– $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$;

– $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$;

– $t_1 \notin t_2 \equiv \neg(t_1 \in t_2)$;

– $\mathbf{F}\ \varphi \equiv$ *true* $\mathbf{U}\ \varphi$;

– $\mathbf{G}\ \varphi \equiv \neg\mathbf{F}\ \neg\varphi$;

– $r \diamondsuit\!\!\rightarrow \varphi \equiv \neg r\ \mid\!\!\rightarrow \neg\varphi$.

– $\exists i \in 1..t.size\ (\varphi) \equiv \neg\forall i \in 1..t.size\ (\neg\varphi)$;

– $\exists v : \tau\ (\varphi) \equiv \neg\forall v : \tau\ (\neg\varphi)$;

– $\forall v \in t(\varphi) \equiv \forall v : \tau\ (x \in t \rightarrow \varphi)$;

– $\exists v \in t(\varphi) \equiv \neg\forall v \in t(\neg\varphi)$;

In the following we use $\varphi[\varphi_2/\varphi_1]$ to denote the TL_D formula obtained by substituting any occurrence of TL_D sub-formula φ_1 of φ with the TL_D formula φ_2.

Formulas in the form $\mathbf{G}\ \varphi$ where φ is a Boolean combination of atomic formulas where *next* does not occur are called *static constraint*s or *invariant*s. Atomic formulas without *next* operators are called *state formula*s. Finally, we denote with TL_{Σ} the language TL_D in the case there are no attribute symbols.

Remark 1 If φ is a state formula, then $\forall v \in t\ (\varphi) \equiv \forall v : \tau\ ((\forall i \in 1..t.\mathit{size}\ (t[i] = v)) \rightarrow \varphi) \equiv \forall i \in 1..t.\mathit{size}\ (\varphi[t[i]/v])$.

4.2 Semantics

TL_D formulas are interpreted over sequences of object models. An object model M consists of a universe $\mathcal{U}$ and of an interpretation $\mathcal{I}$ of the symbols in Σ and of the attribute symbols of D. In particular, every type $\tau \in C \cup \mathit{PT}$ is associated with a non-empty subset of the universe $U_\tau \subseteq \mathcal{U}$, called the domain. For every class $c \in C$, for every attribute $a \in c.A$, $\mathcal{I}(a)$ is a function from U_c to $U_{a.\mathit{Type}}^{a.\mathit{Mult}}$ where $U_{a.\mathit{Type}}^{a.\mathit{Mult}}$ denotes the set of tuples over $U_{a.\mathit{Type}}$ whose size ranges over $a.\mathit{Mult}$.

We consider a first-order Σ-theory $\mathcal{T}$ that constrains the interpretation of the primitive symbols in Σ. We consider only models M that satisfy $\mathcal{T}$ ($M \models \mathcal{T}$). We remark that the $\mathcal{T}$ does not vary over the elements of a sequence of object models. For this reason, symbols such as the operations over the primitive types are rigid, i.e., their interpretation does not change over the different elements of the sequence of object models. On the contrary the value of attributes and other uninterpreted symbols can vary over the different elements of a sequence of object models.

In the following we denote with: ω an infinite sequence of such object models; with ω^i the $i + 1$-th element of the sequence; with $\omega^{i..}$ the suffix sequence $\omega^i, \omega^{i+1}, \ldots$; with U^n, for a set U and for $n \geq 1$, the Cartesian product $\overbrace{U \times \ldots \times U}^{n \text{ times}}$; with $|U|$ the cardinality of a set U; with $|t|$ the size of a tuple t.

Given a term t, an object model M, and an assignment μ to the free variables of t, we define the interpretation $[\![t]\!]_{\langle M,\mu\rangle}$ as follows:

Definition 7 (Interpretation of TL_D static terms)

– $[\![v]\!]_{\langle M,\mu\rangle} = \mu(v)$ *for any variable* v*;*

– $[\![c]\!]_{\langle M,\mu\rangle} = \mathcal{I}(c)$ *for any constant* c*;*

– $[\![f(t_1, \ldots, t_n)]\!]_{\langle M,\mu\rangle} = \mathcal{I}(f)([\![t_1]\!]_{\langle M,\mu\rangle} \cdots [\![t_n]\!]_{\langle M,\mu\rangle})$.

– *If c.a.Mult* $= 1..1$*, and* $\mathcal{I}(a)([\![t]\!]_{\langle M,\mu\rangle}) = \langle q\rangle$ *for some* $q \in U_{c.a.Type}$*, then* $[\![t.a]\!]_{\langle M,\mu\rangle} = q$.

– *If c.a.Mult* $= m..n$*, then* $[\![t.a]\!]_{\langle M,\mu\rangle} = \mathcal{I}(a)([\![t]\!]_{\langle M,\mu\rangle})$ *and* $[\![t.a.size]\!]_{\langle M,\mu\rangle} = |\mathcal{I}(a)([\![t]\!]_{\langle M,\mu\rangle})|$.

– *If* $[\![t]\!] \in U^n_\tau$ *and* $1 \leq [\![i]\!] \leq n$*, then* $[\![t[i]]\!]$ *is the* $[\![i]\!]$*-th element of the sequence* $[\![t]\!]$*; otherwise* $[\![t[i]]\!]$ *is any element in* U_τ.

Definition 8 **(TL_D temporal terms)**

– $[\![t]\!]_{\langle \omega,\mu\rangle} = [\![t]\!]_{\langle \omega^0,\mu\rangle}$ *for any term* t*;*

– $[\![next(t)]\!]_{\langle \omega,\mu\rangle} = [\![t]\!]_{\langle \omega^1,\mu\rangle}$ *for any term* t.

Definition 9 **(TL_D atomic formulas)**

– $\langle \omega, \mu\rangle \models R(t_1, \ldots, t_n)$ *iff* $\mathcal{I}(R)([\![t_1]\!]_{\langle \omega,\mu\rangle} \cdots [\![t_n]\!]_{\langle \omega,\mu\rangle})$ *holds.*

– $\langle \omega, \mu\rangle \models t_1 = t_2$ *iff* $[\![t_1]\!]_{\langle \omega,\mu\rangle} = [\![t_2]\!]_{\langle \omega,\mu\rangle}$.

- $\langle\omega,\mu\rangle \models t_1 \in t_2$ *iff for some* i, $1 \leq i \leq |[\![t_2]\!]_{\langle\omega,\mu\rangle}|$, $[\![t_1]\!]_{\langle\omega,\mu\rangle}$ *is the* i*-th element of* $[\![t_2]\!]_{\langle\omega,\mu\rangle}$.
- $\langle\omega,\mu\rangle \models \neg\varphi_1$ *iff* $\langle\omega,\mu\rangle \not\models \varphi_1$,

  $\langle\omega,\mu\rangle \models \varphi_1 \wedge \varphi_2$ *iff* $\langle\omega,\mu\rangle \models \varphi_1$ *and* $\langle\omega,\mu\rangle \models \varphi_2$.
- $\langle\omega,\mu\rangle \models \forall i \in 1..t.size\ (\varphi)$ *iff for all* j, $1 \leq j \leq [\![t.size]\!]$, $\langle\omega,\mu'\rangle \models \varphi$, *where* $\mu'(v) = \mu(v)$ *for all free variables in* $\forall i \in 1..t.size\ (\varphi)$, *and* $\mu'(i) = j$.
- $\langle\omega,\mu\rangle \models \forall v : \tau\ (\varphi)$ *iff for all* $q \in U_\tau$, $\langle\omega,\mu'\rangle \models \varphi$, *where* $\mu'(v) = \mu(v)$ *for all free variables in* $\forall v : \tau\ (\varphi)$, *and* $\mu'(v) = q$.

Definition 10 (TL_D regular expressions)

- $\langle\omega,\mu\rangle \models^{i..j} \varphi$ *iff* $j = i+1$ *and* $\langle\omega^{i..},\mu\rangle \models \varphi$ *for any atomic formula* φ*;*
- $\langle\omega,\mu\rangle \models^{i..j} \epsilon$ *iff* $i = j$*;*
- $\langle\omega,\mu\rangle \models^{i..j} r*$ *iff* $i = j$ *or there exists* k, $i < k \leq j$, $\langle\omega,\mu\rangle \models^{i..k} r$, $\langle\omega,\mu\rangle \models^{k..j} r*$*;*
- $\langle\omega,\mu\rangle \models^{i..j} r_1; r_2$ *iff there exists* k, $i \leq k \leq j$, $\langle\omega,\mu\rangle \models^{i..k} r_1$, $\langle\omega,\mu\rangle \models^{k..j} r_2$*;*
- $\langle\omega,\mu\rangle \models^{i..j} r_1 : r_2$ *iff there exists* k, $i \leq k < j$, $\langle\omega,\mu\rangle \models^{i..k+1} r_1$, $\langle\omega,\mu\rangle \models^{k..j} r_2$*;*
- $\langle\omega,\mu\rangle \models^{i..j} r_1 \vee r_2$ *iff* $\langle\omega,\mu\rangle \models^{i..j} r_1$ *or* $\langle\omega,\mu\rangle \models^{i..j} r_2$*;*
- $\langle\omega,\mu\rangle \models^{i..j} r_1 \wedge r_2$ *iff* $\langle\omega,\mu\rangle \models^{i..j} r_1$ *and* $\langle\omega,\mu\rangle \models^{i..j} r_2$.

Definition 11 (TL_D formulas)

- $\langle\omega,\mu\rangle \models \neg\varphi_1$ *iff* $\langle\omega,\mu\rangle \not\models \varphi_1$*;*
- $\langle\omega,\mu\rangle \models \varphi_1 \wedge \varphi_2$ *iff* $\langle\omega,\mu\rangle \models \varphi_1$ *and* $\langle\omega,\mu\rangle \models \varphi_2$*;*

– $\langle \omega, \mu \rangle \models X\varphi$ *iff* $\langle \omega^{1..}, \mu \rangle \models \varphi$*;*

– $\langle \omega, \mu \rangle \models \varphi_1 \mathbf{U}\, \varphi_2$ *iff there exists* $i \geq 0$ *such that* $\langle \omega^{i..}, \mu \rangle \models \varphi_2$ *and for all* $0 \leq j < i$ $\langle \omega^{j..}, \mu \rangle \models \varphi_1$*;*

– $\langle \omega, \mu \rangle \models r \;\mapsto \varphi$ *iff for all* $i \geq 0$ *if* $\langle \omega, \mu \rangle \models^{0..i+1} r$ *then* $\langle \omega^{i..}, \mu \rangle \models \varphi$*;*

– $\langle \omega, \mu \rangle \models \forall i \in 1..t.size\ (\varphi)$ *iff for all* j*,* $1 \leq j \leq [\![t.size]\!]$*,* $\langle \omega, \mu' \rangle \models \varphi$*, where* $\mu'(v) = \mu(v)$ *for all free variables in* $\forall i \in 1..t.size\ (\varphi)$*, and* $\mu'(i) = j$*;*

– $\langle \omega, \mu \rangle \models \forall v : \tau\ (\varphi)$ *iff for all* $q \in U_\tau$*,* $\langle \omega, \mu' \rangle \models \varphi$*, where* $\mu'(v) = \mu(v)$ *for all free variables in* $\forall v : \tau\ (\varphi)$*, and* $\mu'(v) = q$*.*

Definition 12 (Satisfiability) *Given a formula* φ*, the satisfiability problem consists of finding a sequence of object models* ω*, and an assignment* μ *to the free variables of* φ*, such that* $\langle \omega, \mu \rangle \models \varphi$*.*

Definition 13 (Equi-Satisfiability) *Given two formulas* φ_1 *and* φ_2*, we say that* φ_1 *and* φ_2 *are equi-satisfiable when* φ_1 *is satisfiable iff* φ_2 *is satisfiable.*

Remark 2 The decidability of the satisfiability problem depends on the theory $\mathcal{T}$. When Σ contains only uninterpreted Boolean constant symbols (thus, with $\mathcal{T}$ empty), TL_Σ turns out to be propositional Linear-time Temporal Logic (LTL) [31] extended with Regular Expressions (RELTL) [27]. Thus, the satisfiability problem can be reduced to a finite-state model checking problem. On the contrary, if the satisfiability problem for the theory $\mathcal{T}$ is undecidable, also the problem for TL_D, which subsumes the set of $\mathcal{T}$ formulas, is undecidable. Finally, note that even if the problem is decidable for $\mathcal{T}$, it might be undecidable for TL_D (see, e.g., [23]).

4.3 Running example

Here we complete the representation of the requirements proposed in the running example via the specification of constraints.

C#1 $\mathbf{G}\ \forall \mathtt{bg} : \mathtt{Balise\_Group}\ (\forall \mathtt{b} \in \mathtt{bg.balises}$
$$(\mathtt{b.internal\_number} \geq 1) \wedge (\mathtt{b.internal\_number} \leq 8))$$

C#2 $\mathbf{G}\ \forall \mathtt{bg} : \mathtt{Balise\_Group}\ (\forall \mathtt{b} \in \mathtt{bg.balises}$
$$(\mathtt{b.bg\_balise\_number} = \mathtt{bg.balises.size}))$$

C#3 $\mathbf{G}\ \forall \mathtt{bg} : \mathtt{Balise\_Group}\ (\forall \mathtt{b} \in \mathtt{bg.balises}\ (\mathtt{b.bg\_id} = \mathtt{bg}))$

C#4 $\mathbf{G}\ \forall \mathtt{bg} : \mathtt{Balise\_Group}\ (\forall \mathtt{b}_1 \in \mathtt{bg.balises}\ (\forall \mathtt{b}_2 \in \mathtt{bg.balises}$
$$(\mathtt{b}_1\mathtt{.relative\_position} < \mathtt{b}_2\mathtt{.relative\_position} \leftrightarrow$$
$$\mathtt{b}_1\mathtt{.internal\_number} < \mathtt{b}_2\mathtt{.internal\_number})))$$

C#5 $\mathbf{G}\ \forall \mathtt{o} : \mathtt{On\_Board\_SS}(\mathtt{o.receive\_linking\_information})$
$$\rightarrow ((\mathtt{o.last\_relevant\_balise\_group\_memorised})\ \mathbf{U}$$
$$(\mathtt{o.received\_coordinate\_system\_RBC})$$

The constraint C#1 (from R#2) expresses that the attribute `internal_number` of a balise associated with a balise group assumes values in the range [1..8]. The constraint C#2 (from R#2) expresses that the attribute `bg_balise_number` of a balise associated with a balise group is exactly the number of balises in the balise group. The constraint C#3 (again from R#2) states that the attribute `bg_id` of a balise associated with a balise group is exactly the identity of the balise group. The constraint C#4 (from R#3) gives a relationship between the physical relative position of two balises in a balise group and their internal number. Finally, constraint C#5 (from R#4.d) states that, if the on-board has not received the linking information, then the Last Relevant Balise Group shall be memorised on board until a coordinate system is received.

5 Formal Analysis and Satisfiability

Given a set of requirements $\Phi = \{\varphi_1, \ldots, \varphi_n\}$, the requirement validation process consists of checking if the requirements are: *consistent*, i.e. they do not contain some contradiction; *not too strict*, i.e. they do allow some desired behavior ψ_d; *not too weak*, i.e. they rule out some undesired behavior ψ_u. Each of these checks can be carried out by solving a satisfiability problem. Consistency checking is performed by solving the satisfiability problem of $\bigwedge_{1 \leq i \leq n} \varphi_i$. The check that the requirements are not too strict is performed by checking whether $\bigwedge_{1 \leq i \leq n} \varphi_i \wedge \psi_d$ is satisfiable. Finally, the check that the requirements are not too weak is performed by checking whether $\bigwedge_{1 \leq i \leq n} \varphi_i \wedge \psi_u$ is unsatisfiable.

Unfortunately, for many underlying theories, the satisfiability problem of TL_D is undecidable (see Remark 2). Nevertheless, we want to keep its expressiveness in order to faithfully represent the informal requirements in the formal language. Thus, we rely on some simplifications that rule out the models with infinite sets of objects. In particular, we assume, first, that the cardinality of attributes of collection type is bounded, second, that the number of objects of some classes is bounded.

In the EuRailCheck project, we asked the user to specify a bound for every class of the formal specification. Exploiting the techniques presented in [12], we can limit this manual operation only to the classes $\tau \in C$ for which a quantifier of the form $\forall v : \tau$ occurs in the formula under analysis. This way, we can automatically find a bound for the remaining classes, we can translate the formula into a Fair Transition System [28] whose accepted language is not empty if the formula is satisfiable, and we can rely on efficient techniques and tools to check for the non-emptiness of the accepted language.

Therefore, the problem that is taken as input by the automatic translation procedure is the following:

Definition 14 (Validation problem) *A validation problem consists of a triple $\langle \varphi, cb, cacb \rangle$, where φ is a TL_D temporal formula, and cb and $cacb$ are natural numbers.*

A solution for the validation problem $\langle \varphi, cb, cacb \rangle$ is a sequence of object models ω satisfying φ such that:

1. *for all classes $c \in C$ with $\forall v : c$ occurring in φ, $|U_c| \leq cb$;*
2. *for all classes $c \in C$, for all attributes a in $c.A$ with multiplicity in the form $m..*$, for all objects $o \in U_c$, $|o.a| \leq cacb$.*

Intuitively, cb represents the bound on the number of objects for each class $c \in C$; and, $cacb$ represents the bound on the cardinality of attributes of type collection.

6 Reduction to Fair Transition Systems

6.1 Reduction procedure

We now outline a procedure which reduces a validation problem into checking whether the language of a *Fair Transition System* (FTS) [28] is not empty: namely, the language of the FTS is not empty iff there exists a solution to the validation problem. The procedure is based on three steps:

1. the TL_D formula is rewritten into an equi-satisfiable one free of quantifiers;
2. the TL_D formula free of quantifiers is further rewritten into an equi-satisfiable one free of attribute symbols;
3. the resulting TL_D formula is encoded into an FTS with standard techniques for the compilation of temporal formulas.

6.2 Removing guarded quantifiers

Given a validation problem $\langle \varphi, cb, cacb \rangle$, we can obtain a quantifier-free formula φ_G, which is satisfiable iff the validation problem has a solution.

In the following, for a given attribute term t, we denote with m_t the maximum between $\max(t.Mult)$ and $cacb$.

We perform the following recursive transformation:

– $\chi(\forall v : \tau(\varphi)) = \bigwedge_{o \in U_\tau}(\chi(\varphi[o/v]))$.
– $\chi(\forall v \in 1..t.size(\varphi)) = \bigwedge_{1 \leq i \leq m_t}(t.size \geq i \rightarrow \chi(\varphi))$.
– $\chi(t_1 \in t_2) = \bigvee_{1 \leq i \leq m_{t_2}}(t_2.size \geq i \wedge t_1 = t_2[i])$.
– $\chi(\neg \varphi_1) = \neg \chi(\varphi_1)$.
– $\chi(\varphi_1 \wedge \varphi_2) = \chi(\varphi_1) \wedge \chi(\varphi_2)$.
– $\chi(r_1 \bowtie r_2) = \chi(r_1) \bowtie \chi(r_2)$, for $\bowtie \in \{;, :, |, \&\&\}$.
– $\chi(r*) = \chi(r)*$.
– $\chi(\mathbf{X}\ \varphi) = \mathbf{X}\ \chi(\varphi)$.
– $\chi(\varphi_1\ \mathbf{U}\ \varphi_2) = \chi(\varphi_2)\ \mathbf{U}\ \chi(\varphi_2)$.

- $\chi(r \mapsto \varphi) = \chi(r) \mapsto \chi(\varphi)$.
- $\chi(\varphi) = \varphi$ otherwise.

Theorem 1 *$\langle \varphi, cb, cacb \rangle$ is satisfiable iff $\chi(\varphi)$ is satisfiable.*

Proof (Sketched proof) Given the assumptions on the validation problem, every quantifier is guarded by a condition with a finite number of models. The quantifiers elimination is therefore straightforward obtained by enumerating these models.

6.3 Finite instantiation

Given the quantifier-free formula φ_G, we can translate the formula into an equi-satisfiable one where all object variables are encoded into a finite domain. More precisely, given a class type c whose domain is not bounded, let n_c be the number of temporal terms of type c that occur in φ_G. For the classes with a bounded domain, let n_c be equal to cb. For each variable v of type c that occurs in φ_G, let us introduce a new variable v_b over the range $[1..n_c]$. For each attribute a of the class c that occurs in φ_G, let us introduce a new function symbol a_b such that: if a is of class type d, then a_b is of type $[1..n_c] \rightarrow [1..n_d]$; if a is of primitive type τ, then a_b is of type $[1..n_c] \rightarrow U_\tau$.

Let φ_b be the result of substituting in φ_G every variable v with v_b, and every attribute expression $c.a$ recursively with $a_b(c)$.

Theorem 2 *φ_G and φ_b are equi-satisfiable.*

Proof Let $\langle \omega, \mu \rangle \models \varphi_G$. For every $i \geq 0$, for every class c, let us consider the set $D_c^i \subseteq U_c$ such that $o \in D_c^i$ if and only if there exists a temporal term t in φ_G

such that $[\![t]\!]_{\langle\omega^i,\mu\rangle} = o$. Let us consider an infinite sequence of injective functions $m_c^i : D_c^i \rightarrow [1..n_c]$, such that for all $i \geq 0$, for all terms t in φ_G of type c, if $[\![t]\!]_{\langle\omega^i,\mu\rangle} = [\![t]\!]_{\langle\omega^{i+1},\mu\rangle} = o \in D_c^i$, then $m_c^i(o) = m_c^{i+1}(o)$. Let us consider a model $\langle\omega_b,\mu_b\rangle$ such that for all $i \geq 0$, for all terms t_b in φ_b of type c, if $[\![t]\!]_{\langle\omega^i,\mu\rangle} = o$ then $[\![t_b]\!]_{\langle\omega_b^i,\mu_b\rangle} = m_c^i(o)$. Then $\langle\omega_b,\mu_b\rangle \models \varphi_b$

Let $\langle\omega_b,\mu_b\rangle \models \varphi_b$. For every $i \geq 0$, for every class c, let us consider an injective function $m_c : [1..n_c] \rightarrow U_c$. Let us consider a model $\langle\omega,\mu\rangle$ such that for all $i \geq 0$, for all terms t in φ_G of type c, if $[\![t_b]\!]_{\langle\omega_b^i,\mu_b\rangle} = j$ then $[\![t]\!]_{\langle\omega^i,\mu\rangle} = m_c(j)$. Then $\langle\omega,\mu\rangle \models \varphi_G$

Note in particular, that if φ (and thus φ_G) contains only Boolean attributes, φ_b is propositional, and we can check its satisfiability by classic finite-state model checking.

6.4 Reduction to Fair Transition Systems Non-Emptiness

FTSs are a symbolic representation of infinite-state systems. We generalize the definition in order to use formulas with symbols in Σ.

Definition 15 (FTS) *A* Fair Transition System *is a tuple $S = \langle\Sigma, V, I, T, F\rangle$, where*

- *Σ is a first-order signature, that defines the state space; the states of the system are defined as the interpretations of the symbols in Σ; we consider only the interpretations that satisfy a given Σ-theory $\mathcal{T}$.*
- *V is a finite set of variables that represent parameters of the system.*

– *I is the initial condition expressed as a TL_Σ state formula.*

– *T is the transition condition expressed as a TL_Σ atomic formula.*

– *F is the set of fairness conditions, each condition expressed as a TL_Σ state formula.*

Given an infinite sequence ω of Σ-interpretations and an assignment μ to the parameters in V, S accepts $\langle \omega, \mu \rangle$ iff $\langle \omega^{i..}, \mu \rangle \models T$ for every $i \geq 0$, $\langle \omega^0, \mu \rangle \models I$, and, for all $\psi \in F$, there exist infinitely many i, such that $\langle \omega^i, \mu \rangle \models \psi$. The language $\mathcal{L}(S)$ is defined as the set of pairs $\langle \omega, \mu \rangle$ accepted by S.

In the case of RELTL, there exist efficient techniques to convert a temporal formula φ into an equivalent FTS S_φ (see, e.g., [13]). Formally,

Theorem 3 *Given a TL_Σ formula φ that contains only uninterpreted Boolean constants and Boolean variables V, there exists an FTS $S_\varphi = \langle \Sigma, V, I, T, F \rangle$ such that $\langle \omega, \mu \rangle \models \varphi$ iff $\langle \omega, \mu \rangle \in \mathcal{L}(S_\varphi)$.*

Thus, the check for the satisfiability of φ can be performed verifying that $\mathcal{L}(S_\varphi) \neq \emptyset$.

As for the general case, we solve the satisfiability problem of a temporal formula φ as follows. Let $A(\varphi)$ the set of atomic sub-formulas of φ. First, for every atomic formula ψ in $A(\varphi)$, we introduce a new Boolean uninterpreted constant a_ψ. Second, we consider the Boolean abstraction φ^A of φ where every atomic formula ψ has been substituted with the corresponding Boolean constant a_ψ.

Theorem 4 *φ is equi-satisfiable to $\varphi^A \wedge \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\,(a_\psi \leftrightarrow \psi)$.*

Proof Suppose $\langle \omega, \mu \rangle \models \varphi$, where $\mathcal{I}^i$ is the interpretation of ω^i for all $i \geq 0$. Let us consider a new sequence of models ω' such that, for all $i \geq 0$, the interpretation $\mathcal{I}'^i$ of ω'^i is defined as follows: $\mathcal{I}'^i(s) = \mathcal{I}^i(s)$ for all symbols occurring in φ, while $\mathcal{I}'^i(a_\psi) = \mathit{true}$ iff $\langle \omega'^i, \mu \rangle \models \psi$. Then $\langle \omega', \mu \rangle \models \varphi^A \wedge \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$.

Suppose $\langle \omega, \mu \rangle \models \varphi^A \wedge \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$ Note that, for all $i \geq 0$, $\langle \omega^i, \mu \rangle \models a_\psi$ iff $\langle \omega^i, \mu \rangle \models \psi$. Thus $\langle \omega, \mu \rangle \models \varphi$.

Theorem 5 *If φ^A is equivalent to the FTS S_{φ^A}, then $\varphi^A \wedge \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$ is equivalent to the FTS S_φ obtained by adding $\bigwedge_{\psi \in A(\varphi)}(a_\psi \leftrightarrow \psi)$ to the transition condition of S_{φ^A}.*

Proof Suppose $\langle \omega, \mu \rangle \models \varphi^A \wedge \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$. In particular, $\langle \omega, \mu \rangle \models \varphi^A$. Thus, $\langle \omega, \mu \rangle \in \mathcal{L}(S_{\varphi^A})$. Moreover, $\langle \omega, \mu \rangle \models \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$. Then, for all $i \geq 0$, $\langle \omega^{i..}, \mu \rangle \models \bigwedge_{\psi \in A(\varphi)}(a_\psi \leftrightarrow \psi)$. Thus, $\langle \omega, \mu \rangle \in \mathcal{L}(S_\varphi)$.

Suppose $\langle \omega, \mu \rangle \in \mathcal{L}(S_\varphi)$. Thus, $\langle \omega, \mu \rangle \in \mathcal{L}(S_{\varphi^A})$ and $\langle \omega, \mu \rangle \models \varphi^A$. Moreover, for all $i \geq 0$, $\langle \omega^{i..}, \mu \rangle \models \bigwedge_{\psi \in A(\varphi)}(a_\psi \leftrightarrow \psi)$. Then $\langle \omega, \mu \rangle \models \varphi^A \wedge \bigwedge_{\psi \in A(\varphi)} \mathbf{G}\ (a_\psi \leftrightarrow \psi)$.

Corollary 1 *φ is satisfiable iff $\mathcal{L}(S_\varphi) \neq \emptyset$.*

Therefore, we can check the satisfiability of φ, by checking the non-emptiness of the language accepted by S_φ.

6.5 Running Example

We first consider the consistency problem of the requirements of the running example. If we consider for example C#1 (see Section 4.3), the quantifier-free corresponding formula would be:

C#1' $\mathbf{G} \bigwedge_{\mathtt{bg} \in U_{\mathtt{Balise\_Group}}} (\bigwedge_{\mathtt{i} \in 1..8} (\mathtt{i} \leq \mathtt{bg.balises.size} \rightarrow$
$(\mathtt{bg.balises[i].bg\_balise\_number} \geq 1) \wedge$
$(\mathtt{bg.balises[i].bg\_balise\_number} \leq 8)))$

In the constraints introduced in Section 4.3, the classes for which there are no occurring quantifiers are `RBC`, `Linking_Information`, `Telegram`, and `Balise`. For the first three classes, there is no term of the corresponding type in the constraints. Thus, we can consider an empty set of objects for these classes. For the class `Balise`, we note that the constraint C#1' contains $8 \times |U_{\mathtt{Balise\_Group}}|$ terms of type `Balise`. Constraints C#2-5 do not add new terms of type `Balise`. Thus, we can encode the objects of type `Balise` with $1..8 \times |U_{\mathtt{Balise\_Group}}|$.

7 Experimental evaluation

7.1 Implementation

The reduction procedure described in previous sections was implemented on top of the NuSMV symbolic model checker [8]. The internal functionalities provided by NuSMV were used as a basis for the implementation of the various steps of the procedure (the elimination of the guarded quantifiers discussed in Section 6.2, the finite instantiation step discussed in Section 6.3, and the conversion into an FTS of the resulting formula discussed in Section 6.4). The analysis of the emptiness of the language of the resulting FTS was carried out by means of an extended version of NuSMV, that provides abstraction refinement functionalities [7] and has been interfaced with the MathSAT [5] Satisfiability Modulo Theory (SMT) solver. In the following, we briefly present the Counterexample-Guided Abstraction Refinement and Bounded Model Checking techniques that we used to check whether the language of the FTS resulting from the reduction is empty.

Counterexample-Guided Abstraction Refinement (CEGAR) We check the non-emptiness of S_φ by means of predicate abstraction. We adopt a CEGAR loop [14], where the abstraction generation and refinement are completely automated. The loop consists of four phases (Fig. 3 provides a pictorial representation of the CEGAR loop):

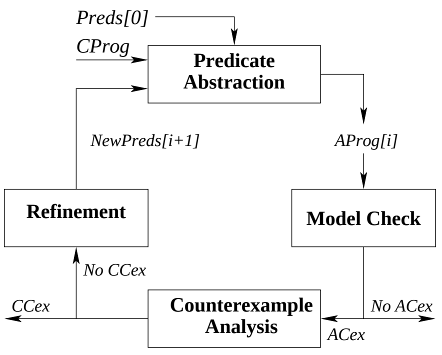

Fig. 3 The CEGAR loop.

– *abstraction*, where the abstract system is built according to a given set of predicates;

- *verification*, where the non-emptiness of the language of the abstract system is checked; if the language is empty, it can be concluded that also the concrete system has an empty language; otherwise, an infinite trace is produced;
- *simulation*: if the verification produces a trace, the simulation checks whether it is realistic by simulating it on the concrete system; if the trace can be simulated in the concrete system, it is reported as a real witness of the satisfiability of the formula;
- *refinement*: if the simulation cannot find a concrete trace corresponding to the abstract one, the refinement discovers new predicates that, once added to the abstraction, are sufficient to rule out the unrealistic path.

The abstract FTS $S_a = \langle \Sigma, V_p, I_a, T_a, F_a \rangle$ of an FTS $S = \langle \Sigma, V, I, T, F \rangle$ with regards to the set of predicates P at each iteration can be computed with the following Boolean formulas:

$$I_a(\hat{V}) = \exists V (I(V) \wedge \bigwedge_{p \in P} (\hat{v}_p \leftrightarrow p(V)))$$
$$T_a(\hat{V}, \hat{V}') = \exists V \exists V' (T(V, V') \wedge \bigwedge_{p \in P} (\hat{v}_p \leftrightarrow p(V) \wedge \hat{v}'_p \leftrightarrow p(V')))$$
$$F_a = \{ f_a(\hat{V}) | f_a(\hat{V}) = \exists V (f(V) \wedge \bigwedge_{p \in P} (\hat{v}_p \leftrightarrow p(V))) \text{ for } f \in F \}$$

The quantifiers are removed by passing each formula to a decision procedure, and enumerating the satisfying assignments to the abstract variables $\hat{v}_p \in \hat{V}, \hat{v}'_p \in \hat{V}'$ [17].

As in [26] and [7], I, T and each $f \in F$ are first-order formulas and the computation of all solutions is carried out by a SAT-modulo-Theories (SMT) solver.

In the abstract space we search for a lasso-shaped trace $\hat{s}_0, \ldots, \hat{s}_k$ with $\hat{s}_l = \hat{s}_k$ for $l < k$ such that for all $f_a \in F_a$ there exists a state $\hat{s}_i$ with $l \leq i \leq k$ such that

$\hat{s}_i \in f_a$ (i.e. the f_a fairness condition is satisfied in $\hat{s}_i$). This trace can be generated by any standard model checking techniques [15].

To check whether the abstract trace corresponds to a concrete one, we encode the simulation of the abstract trace on the concrete system into a Bounded Model Checking (BMC) problem [3], as proposed by [16]. Since the abstract trace is lasso-shaped, i.e. it is given by the abstract states $\hat{s}_0, \dots, \hat{s}_k$ with $\hat{s}_l = \hat{s}_k$ for $l < k$, we then generate the SMT formula:

$$I(V_0) \wedge \bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \wedge \bigwedge_{0 \leq i \leq k} \hat{s}(\hat{V}_i) \wedge \bigwedge_{0 \leq i \leq k} \bigwedge_{v \in \hat{V}} (\hat{v}_i \leftrightarrow p(V_i)) \wedge \bigwedge_{v \in V} v_l = v_k \quad (1)$$

If the formula is not satisfiable, we refine the abstraction, otherwise we have found a concrete trace that witnesses the non emptiness of the language accepted by the concrete FTS. As noted in [32], this concretization step is sound but not complete: we are fixing a specific lasso shape, so it may happen that the above formula is not satisfiable, but the counterexample is concretizable.

The initial set of predicates is empty. The new predicates discovered during the refinement are automatically generated by analyzing the proof of the unsatisfiability of (1). To this purpose, for instance it is possible to use the interpolation-based techniques described in [9].

Bounded Model Checking via SMT Although incomplete, the CEGAR loop is often able to prove the inconsistency of a property φ by finding the right abstraction. On the opposite, if we aim at looking for a witness of the formula's satisfiability, a more effective procedure is to look for a lasso-shape trace of length up to a given bound. In this case, we encode the simulation of such trace in the concrete FTS with the following SMT formula:

$$I(V_0) \wedge \bigwedge_{0 \leq i < k} T(V_i, V_{i+1}) \wedge \bigvee_{0 \leq l < k} \bigwedge_{v \in V} v_l = v_k \wedge \bigwedge_{\psi \in F} \bigvee_{l \leq h < k} \psi(V_h) \qquad (2)$$

If the above formula is satisfiable, then we have found a concrete trace witnessing the non emptiness of the language of the concrete FTS. If the formula is unsatisfiable, we increment the k in the above formula and we re-submit the resulting formula to an SMT solver.

7.2 Case study

In this work we refer to a subset of the ETCS specifications. In particular, we refer to 25 requirements from the subset 3 of the SRS 026 [20]. These requirements are related to the fragments 3.4 and 3.16, that describe the structure and behavior of the balises and balise groups (including the requirements of the running example), and to the fragment 3.8, that is focused on the description of the structure of the *Movement Authority*, i.e., the part of the trackside a given train is authorized to move by the ground control system. The 25 requirements have been formalized in 10 classes, each one with an average of 8 attributes, and 22 constraints. We remark that this subset contains part of the requirements we analyzed and validated during the EuRailCheck project.

7.3 Results

We ran the experiments on a 2.20GHz Intel Core2 Duo Laptop equipped with 2GB of memory running Linux version 2.6.24. We fixed a memory limit of 1Gb and a

		BMC k = 5			BMC k = 10			BMC k = 15			CEGAR		
Name:	$\#r$,$\#i$,$\#b$	Sat	Mem (MB)	Time (sec)	Sat	Mem (MB)	Time (sec)	Sat	Mem (MB)	Time (sec)	Sat	Mem (MB)	Time (sec)
SAT	408,132,200	Y	116.2	31.65	Y	170.6	75.50	Y	228.8	131.14	-	-	-
SCEN_1	440,132,200	Y	118.8	42.49	Y	174.0	63.74	Y	234.1	197.27	-	-	-
SCEN_2	440,132,232	Y	120.6	44.97	Y	175.9	85.71	Y	236.1	119.06	-	-	-
SCEN_3	442,132,236	Y	121.9	32.79	Y	180.2	69.07	Y	239.9	105.44	-	-	-
PROP	408,132,200	-	-	T.O.	-	-	-	-	-	-	N	94.0	3.03

Table 2 The results of the experimental evaluation.

CPU limit of 10min. All the data and binaries necessary to reproduce the results here presented are available at `http://es.fbk.eu/people/tonetta/tests/sosym09/`.

We checked whether the specification is consistent, i.e. if it is satisfiable (SAT). Then, we validated the formalization with 3 different scenarios (SCEN_{1,2,3}), checking the satisfiability of the conjunction of the formula representing the scenario and the whole specification. Finally, we checked a property (PROP), proving that the conjunction of the specification with the negation of the formula is unsatisfiable. For each case we considered the validation problem with $cb = 4$, $cacb = 10$. We asked the tool to generate witness traces of different increasing lengths k (5, 10, and 15 respectively) using BMC with SMT techniques, and when the BMC approach was not able to conclude, we tried with the CEGAR approach. We obtained the results reported in Table 2. In the table we also report the size, in terms of number of variables of the FTS we submit to the underlying verification tool. (We use $\#r$,$\#i$,$\#b$ with the meaning, r Real variables, i Integer variables, and b Boolean variables.) We use T.O. to report that the tool was not able to find a solution in the given time limit. We were able to generate witnesses for the satisfiability and for the compatibility with the 3 considered scenarios using BMC with SMT techniques quite efficiently. For this reason, we have not tried to use the more expensive CEGAR approach (that it is more suitable to prove unsatisfiability of the formula). While checking the PROP problem, BMC techniques reached time out with bound $k = 5$. We thus tried to prove with CEGAR whether the property holds. In a few seconds, CEGAR concluded that the language of the abstract FTS is empty, thus showing that the property holds. The abstraction refinement was completely automatic and no manual intervention was necessary. This demonstrates that the different checking techniques are able to complement each other.

7.4 Discussion on scalability

The results show that the validation is feasible. The performance of the validation engine depends on the number of considered requirements. Nevertheless, the property-based approach, by formalizing every requirement with a different formula, allows us to partition the requirements in different sets controlling the complexity of the analysis. Moreover, the checks based on BMC depend on the length of the shortest trace satisfying the requirements; while the checks based on CEGAR depend on the predicates necessary to prove the inconsistency. In our experience, most of time, the problems are satisfiable and the length of witnessing traces is sufficiently small.

8 Related Work

Several attempts to the formal specification and validation of requirements have been proposed. The works that aim at solving problems similar to the ones tackled in this paper are Alloy [25], Formal Tropos [22] and OCL [30].

Alloy [25] is a language for describing structural properties of a system relying on a small kernel formal language based on the subset of Z [33] that allows for object modeling. An Alloy specification consists of basic structures representing classes together with constraints and operations describing how the structures change dynamically. Alloy only allows to specify attributes belonging to finite domains (no Reals or Integers). Thus, it would have been impossible to model exactly the Train position of the running example presented in this paper in Alloy, and some manual abstraction would be necessary. Although Alloy supports the "next" operator ("prime" operator), that allow to specify the temporal evolution of a given object, it does not allow to express properties using LTL and regular expressions (at the basis of the logic presented in this paper). Thus, it is limited to state or transition invariants, and it does not allow to specify fairness conditions that are crucial in our context. Similarly to the approach proposed in this paper, Alloy supports two kinds of analysis: simulation and checking. In simulation, the consistency of an invariant or operation is demonstrated by generating witness (a state or a transition). In checking, a consequence of the specification is tested by attempting to generate a counterexample of a user specified length. Alloy, in order to perform the verification, requires the user to specify the bounds on the maximum number of class instances and a limit for the counterexample/witness length.

Thus, it is only able to prove/disprove a property for the given bounds and for the given length. The approach presented in this paper overcomes these problems with the automatic finite instantiation that preserves satisfiability and with a reduction to known verification techniques that allow to prove/disprove a property regardless of any bound on the witness/counterexample length.

Tropos [4, 34] is an agent-oriented software engineering methodology which provides a visual modelling language that can be used to define an informal specification, allowing to model intentional and social concepts, such as those of actor, goal, and social relationships between actors. Formal Tropos (FT) [22] extends a Tropos specification with annotations that characterize the valid behaviors of the model. A FT specification consists of a sequence of class declarations such as actors, goals, and dependencies. Each declaration associates a set of attributes to the class. The temporal behavior of the instances is specified by means of temporal constraints expressed in a typed first-order LTL. FT is similar in spirit to the approach proposed in this paper. The difference between the proposed approach and FT are in the expressiveness of the formalization language. FT is limited to LTL temporal operators, while this approach allows to express constraints and properties with a logic that mixes LTL with regular expression (which is more expressive than LTL). FT, like Alloy, requires for the formal analysis the specification of bounds on the maximum number of class instances, and can only deal with finite domain class attributes and constraints over such finite domain values. Our novel approach does not require the specification of the bounds on the number of class instances, and allows for the use of class attributes of infinite range like e.g. Integers and Reals.

OCL [30] is a formal language developed as a business modelling language to express additional constraints about the objects in an UML model. It is a pure specification language, so an OCL expression is guaranteed to be without side effects on the model. This means that the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to specify a state change. In fact, in a post-condition, and only in this context, the expression can refer to values for each property of an object (via the operator $@pre$) at two different time instants: the value of a property at the start of the operation or method and the value of a property upon completion of the operation or method. This allows to describe and to predicate on the temporal evolution of a property. Similarly to Alloy, OCL cannot express some temporal properties, such as fairness. In OCL no commitment is done on the possibility to execute analysis, such as model checking, moreover, as a general observation, the language is not decidable.

As far as the combination of temporal with first-order state formulas logic is concerned, we remark that it was proposed by Manna and Pnueli in [28,29] as specification language for reactive systems. Many works studied the decidability of particular fragments of the logic (cf., e.g., [24,18,23]). We also considered a fragment of the language, by restricting temporal quantifications to the guarded quantifiers over bounded sets of objects and to implicit freezing quantifiers that relate next and current values in the transition expressions. Rather than focusing on which are the decidable theories we can use in the state formulas, our work focus more on object models, their specification and validation.

9 Conclusions

In this paper, we have proposed a formalism for the representation and the analysis of requirements. The formalism builds on class diagrams, and combines fragments of first order logic (to describe rich data and relationships between attributes and entities) with temporal operators (to describe the evolution of the scenarios). We proposed a procedure to check the satisfiability based on the finite domain encoding, on the reduction to language non-emptiness for FTS, and on the application of verification techniques based on SMT and abstraction-refinement. We implemented the procedure within an extended version of the NuSMV model checker. The language has been used by railway experts within the EuRailCheck project funded by the European Railway Agency, with the purpose of formalizing and validating a significant subset of the ETCS specification. We consider as case study some paradigmatic requirements and we prove that the specification is satisfiable, that few desired scenarios are allowed, and that a property holds.

In the future, we plan to extend the expressiveness of the formalism to encompass richer data. We will also investigate enhancements in the verification engine, along the lines outlined in [10], and to more aggressive (problem-specific) abstraction techniques. Finally, we will investigate issues related to contract-based specifications, where only a part of the scenario is considered to be controllable, and to address problems beyond satisfiability, such as realizability and synthesis.

References

1. IEEE Standard for Property Specification Language (PSL). *IEEE Std 1850-2005*, 2005.
2. IEEE Standard for System Verilog – Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2005*, 2005.
3. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, pages 193–207, 1999.
4. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
5. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. The MathSAT 4 SMT Solver. In *CAV*, pages 299–303, 2008.
6. D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M.Y. Vardi. Regular Vacuity. In *CHARME*, pages 191–206, 2005.
7. R. Cavada, A. Cimatti, A. Franzén, K. Kalyanasundaram, M. Roveri, and R. K. Shyamasundar. Computing predicate abstractions by integrating BDDs and SMT solvers. In *FMCAD*, pages 69–76. IEEE, 2007.
8. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.
9. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. In *TACAS*, pages 397–412, 2008.
10. A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean abstraction for temporal logic satisfiability. In *CAV*, volume 4590 of *LNCS*, pages 532–546. Springer, 2007.
11. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From Informal Requirements to Property-Driven Formal Validation. In *FMICS*, LNCS, L'Aquila, Italy, sep 2008. Springer.

12. A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Object models with temporal constraints. In *SEFM*, pages 249–258. IEEE Computer Society, 2008.
13. A. Cimatti, M. Roveri, and S. Tonetta. PSL Symbolic Compilation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(10):1737–1750, 2008.
14. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV*, pages 154–169, 2000.
15. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, London, England, 1999. ISBN 0-262-03270-7.
16. E. M. Clarke, A. Gupta, J.H. Kukula, and O. Strichman. SAT Based Abstraction-Refinement Using ILP and Machine Learning Techniques. In *CAV*, pages 265–279, 2002.
17. E. M. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate Abstraction of ANSI-C Programs Using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.
18. S. Demri, R. Lazic, and D. Nowak. On the freeze quantifier in Constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.
19. European train control system – home page. `http://www.era.europa.eu/core/ertms/Pages/FirstETCSSRS300.aspx`.
20. System Requirements Specification - ETCS Subset 026 v230, 2006.
21. Formal Verification of ETCS specifications: EuRailCheck. `http://www.era.europa.eu/core/ertms/Pages/Feasibility_Study.aspx`, and `http://es.fbk.eu/events/formal-etcs`.
22. A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.
23. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems. In *CADE*, pages 362–378,

2007.

24. I. M. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragment of first-order temporal logics. *Ann. Pure Appl. Logic*, 106(1-3):85–134, 2000.
25. D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.
26. S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In *CAV*, LNCS, pages 424–437. Springer, 2006.
27. M. Lange. Linear Time Logics Around PSL: Complexity, Expressiveness, and a Little Bit of Succinctness. In *CONCUR*, pages 90–104, 2007.
28. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer, 1992.
29. Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.
30. OMG. *Object Constraint Language: OMG available specification Version 2.0*, 2006.
31. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.
32. R. Sebastiani, S. Tonetta, and M.Y. Vardi. Property-Driven Partitioning for Abstraction Refinement. In *TACAS*, pages 389–404, 2007.
33. J. M. Spivey. *The Z Notation: a reference manual, 2nd edition*. Prentice Hall, 1992.
34. A. Susi, A. Perini, P. Giorgini, and J. Mylopoulos. The Tropos Metamodel and its Use. *Informatica*, 29(4):401–408, 2005.
35. UML Version 2.1.2. `http://www.omg.org/spec/UML/2.1.2/`.