WELL-FORMED PETRI NET BASED PATTERNS FOR MODELING LOGIC CONTROLLERS FOR AUTONOMOUS TRAINS

Yuchen Xie, Manel Khlif-Bouassida, Armand Toguyéni

Centrale Lille, CRIStAL, UMR 9189
59650 Villeneuve d'Ascq, France
Univ. Lille Nord de France, F-59650, Lille, France

{yuchen.xie, manel.khlif-bouassida, armand.toguyeni}@centralelille.fr

ABSTRACT
The automation and the adoption of ERTMS (The European Rail Traffic Management System) are two solutions for railway systems to satisfy the necessity of increasing the capacity of railway lines and enhancing their safety. In this context, this study is to be part of the contribution to a methodology allowing the development of discrete event controllers of autonomous train control system needed for railway automation. This article emphasizes the modeling stage using Colored Petri Nets (CPN) and its extensions. While modeling, both the railway requirements and the necessity to formally verify some crucial properties (e.g., collision-free system) have been taken into account. By proposing several modeling patterns based on Well-Formed Petri Nets (WFN), we solve several technical problems of modeling railway train control system and similar complex systems, making it possible to construct reducible and analyzable models, before being formally verified.

Keywords: Railway System, Autonomous Train Control, DES Modeling, Colored Petri Nets

1. INTRODUCTION

The development of autonomous trains logic controllers has become a priority in railway control system, in order to increase its safety, and to make railway system more competitive with regard to the other means of transportation.

This study is part of a methodology allowing the systematic and rigorous development of logic controllers necessary for railway automation. The methodology we finally develop should formally model the control functions and make it possible to verify essential properties (e.g., safety) of an automated system. The ultimate goal of this methodology is to generate, by model transformation, the code of these functions, which could be implemented on the computers of the ground infrastructure and on the embedded controllers in the trains. This paper concerns the modeling stage. One problematic situation is to deal with the compromise between the modeling power of the selected modeling approaches and the possibility of making formal verification. In this study, we decide to use Well-Formed Petri Nets (WFN) as the modeling formalism to benefit from all its advantages, and we propose several WFN modeling patterns and techniques suitable with the complexity of railway systems.

The paper is structured as follows. In the second section, we give a state-of-the-art of railway system modeling using Colored Petri Net (CPN). In the third section, we discuss the tradeoff between the modeling power and the verification capacity of CPN and WFN approaches, in order to finally justify the reason why we choose WFN as our modeling tool for autonomous train control system. Section 4 presents a railway system structure and some main functions used in this paper; several constrains and assumptions in the modeling stage are also given in section 4. In section 5, we present a brief introduction about the main formalism and characteristics of WFN. In section 6, we propose certain modeling patterns as solutions to some main modeling problems of a railway control system. In section 7, we illustrate the use of these patterns by a case study of automation of railway system. Finally, we end up with conclusions and perspectives of this work in section 8.

2. STATE OF THE ART

For about half a century, Petri Nets (PN) have been used to model concurrent and complex systems. Among its numerous extensions, CPN is the most widely used formalism incorporating data, hierarchy, and time (van der Aalst et al. 2013). This section summarizes some research works using CPN to model and analyze railway control system.

In (Janczura 1999), a whole process of modeling and analyzing a railway network is proposed using CPN. The network considers two types of trains (i.e., express and normal) which move in the same direction. A safety property (each block in the railway line can only be occupied by exactly one train or empty) and four operational properties are analyzed. However, this thesis report only considers a quite simple model and does not respect the ERTMS/ETCS standard.

In (Jansen et al. 1998; Meyer zu Hörste 1999), a CPN hierarchical framework is proposed to model ERTMS/ETCS (mainly in level-2). Several generic modeling paradigms and techniques (e.g., distributed modeling, communication between separate CPNs, synchronization, etc.) are created to build their CPN models and formal methods can be used to analyze these models. This study mainly focus on the hierarchical and structural problems of modeling ETCS specifications instead of the implementation of concrete functional models.

A summary of Petri Nets models of railway stations is given in (Žarnay 2004). Different models are divided into four levels (i.e., technical equipment level, movement level, train processing level and decision-making level) according to their different objectives and different abstract levels.

CPN Tools is a tool for editing, simulating and analyzing CPNs, which is first introduced in (Ratzer et al. 2003; Jensen et al. 2007). After its wide application, more CPN models are proposed by benefiting from the features of this powerful tool.

In (van der Aalst et al. 2013), several CPN design patterns and strategies are proposed using CPN Tools, showing some solutions to several typical design problems in terms of modeling complex processes.

In (Vanit-Anunchai 2009; Vanit-Anunchai 2010; Vanit-Anunchai 2014), railway interlocking table models are proposed using CPN Tools. As a main advantage of these models, the general CPN structure proposed can be reused regardless of variable structures and sizes of railway systems. While, these models store too many data (e.g., geographic information) in colored tokens and the behavior of the models are greatly affected by the data instead of the structure of model. In this case, although we can do some test with the simulation function in CPN Tools, a formal verification is rather difficult to performed on this kind of CPN models.

In our previous work (Xie et al. 2016), we have proposed discrete controller models based on High-level Petri Nets supported by CPN Tools. Our research concentrates on the whole railway system (i.e., both the train controller and the trackside part). In this previous work, these High-level Petri Nets are unconstrained CPNs, allowing a powerful modeling ability of the systems. CPN Tools also offers extra extensions with ML Language to enhance its modeling power, e.g., the possibility of defining a "list" datatype. However, one pays for this capacity of modeling because there are no really, tools or efficient methods, to help analyze and verify these unconstrained CPNs with extensions.

3. COMPARISON OF CPN AND WFN APPROACHES OF MODELING TRAIN CONTROL SYSTEMS

Figure 1 compares the CPN and WFN approaches of modeling the railway system and the possible analyzing methods applicable to the models.

As shown in the left part of Figure 1, the most direct way of analyzing a CPN is to generate its reachability graph which enables to check the required properties. Although it is possible in theory, the application of this method is generally limited by combinatorial explosion problems.

One way to combat this combinatorial explosion would be to reduce the initial CPN model before constructing the reachability graph. However, there is very little work proposing reduction rules applicable to CPNs and each of them has their own applicable constrains. For example, the reduction rules proposed in (Esparza & Hoffmann 2016) are only applicable to free choice workflow nets with an objective of maintaining the soundness property.

In more general cases, the only solution available is to first go through an unfolding operation of the CPN before the application of some existing reduction rules of ordinary PNs (Berthelot & Lri-Iie 1986; Murata 1989). However, in this case, for a complex system e.g. railway system, one is confronted with a combinatorial explosion problem in the unfolding operation.

To solve the problems above, some High-level Petri Nets with constrains are proposed, among which the Well-Formed Petri Nets (Chiola et al. 1991) are of interest to us. It is proved that WFN have the same expression power as CPN (Diaz 2013), which means, every CPN can be transformed into a WFN with the same basic structure, same color domains (possibly partitioned in static subclasses), equivalent arc labeling, and the possible addition of transition predicates (Chiola et al. 1991). It implies that WFNs have at least an equal modeling power compared to general (unconstrained) CPNs defined in (Jensen 1981).

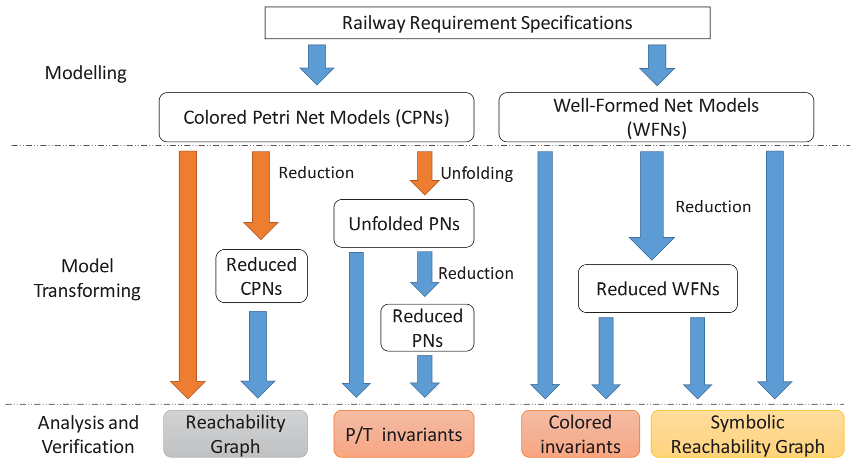

Figure 1 Different Ways of Analysis of Colored Nets

The right part of Figure 1 describes the possible analysis methods applicable to WFN models. To avoid the combinatorial explosion, several reduction rules could first be applied, e.g. the reduction rules in (Haddad 1991). Several reduction rules may also need a calculation of colored invariants (Couvreur & Martínez 1991). Then the models can be analyzed with help of these invariants and/or by building a symbolic reachability graph (Chiola et al. 1991) which will greatly reduce the size of the reachability graph.
As our final objective is to propose appropriate Petri Net patterns whose properties can be checked before the models are implemented, this paper proposes the use of WFN instead of CPN for modeling autonomous railway control systems, in order to benefit from the advantages of analyzing a WFN model.

4. RAILWAY SYSTEM BASIC AND CONTEXT

This study concerns the management of multiple trains in a railway line based on Movement Authorities (MA, permission for a train to move to a specific location with supervision of its speed) generated by trackside infrastructure. We first present the background of the railway models.

4.1. Railway Lines and Blocks

Railway lines are connections of different railway stations. Normally, a single railway line has a fixed direction and all the trains in this railway line run in this direction. A railway line is divided into numerous blocks. Blocks are used to avoid train collisions, ensuring the safe and efficient operations of railway systems.

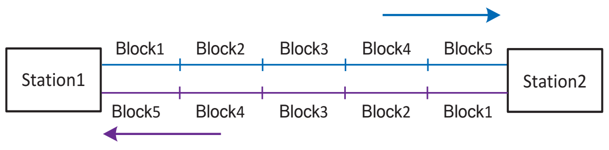

Figure 2 Railway Lines and Blocks

Figure 2 shows an example of lines decomposed into blocks. The railway line from Station 1 to Station 2 and another railway line from Station 2 to Station 1 are divided into several blocks respectively (for simplicity, Figure 2 represents each railway line with 5 blocks). For safety reasons, each block must contain no more than one train. Thus, only after the train occupying the current block (the block is said to be "occupied") has left (the block is then "clear"), another train is authorized to enter this block.

4.2. ETCS-2 Based Train Management in Railway Lines

European railway systems are nowadays equipped with the ERTMS and the European Train Control System (ETCS).
ETCS is specified in four different levels (level 0-3). Currently, ERTMS/ETCS level 2 (ETCS-2) has been put into use on several high-speed railway lines in Europe, which uses Eurobalise to help train locating and uses continuous radio transmission GSM-R (Global System for Mobile Communications - Railway) for data exchanges between trackside infrastructures and onboard equipment. Our study is based on the infrastructure of ETCS-2. Figure 3 illustrates the main functions of train management offered by ETCS-2.

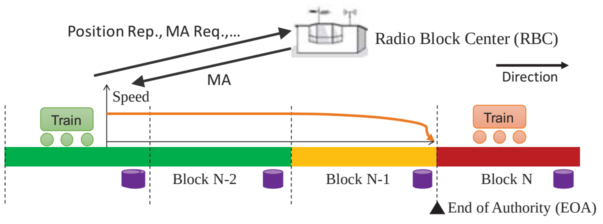

Figure 3 ETCS-2 Based Train Management

Trackside: Radio Block Center (RBC) provides trains with Movement Authorities (MA), taking into account the positions of corresponding trains, signals and switch states as well as the physical line configuration (slopes, curves, etc.);
Onboard: Each train regularly sends its position to RBC and receives MA from RBC. The onboard equipment calculates a speed profile considering the End of Authority (EOA), which is the last block in the MA, and the train characteristics (mass, length, etc.).

4.3. System Simplifications and Assumptions

This study considers a set of simplifying assumptions to manage multiple trains in a railway line. The aim of these simplifications is to reduce the complexity of the models so that the models could be represented in a limited number of pages. The principle assumptions are:

1. Our model does not take into account the length of a train. We only care whether a train occupies a block.
2. The MA message is reduced to the list of blocks that are reserved and assigned to a train. The exact speed limit and the other parameters in a MA are not considered here. However, we assume that a train can always stop at its EOA.
3. A single RBC manages all the trains in the same railway line between two stations. This means that the RBC handover function is beyond this study. The control of railway node/station is not considered.
4. In this paper, a "railway line" has a fixed operation direction and is linked with only 2 stations: the departure and the arrival. The overtaking is not considered. This assumption simplifies the operations we propose later in the paper by always maintaining the same order of the trains as they enter in this railway line.
5. Each time a train enters in a new block, we assume that it receives its current position from a Eurobalise and then sends a position report to the corresponding RBC, instead of considering the specified report format according to ERTMS/ETCS-2 standard.

6. Once a RBC receives a position report from a train, it updates the train's location in its database. The RBC also considers the location report as a MA request. Consequently, it generates a MA response to the train according to the following principle: if the train is the first one in the railway line (there is no preceding train until the end block), its EOA is set to the end block in this railway line, otherwise the EOA is set to the block next to the one occupied by the preceding train.

5. WELL-FORMED PETRI NETS

5.1. Well-Formed Petri Net and its formalism

Well-Formed Petri Nets (WFN) are Colored Petri Nets (Jensen 1981) that satisfy a set of syntactical constraints. In this paper, we only introduce the main features of WFN. A complete formal definition can be found in (Chiola et al. 1991).

5.1.1. WFN Color Classes and Color Domains

A color class can be ordered or unordered, and can be divided into static subclasses. A color class defines the same nature of the tokens of this type. When a color class comprise of several static subclasses, the colors within each static subclass share some similar potential behaviors (batch operation, symmetry, etc.).

A color domain is a Cartesian product of color classes. A neutral color is noted as ε, allowing to define uncolored places or transitions.

Each place and each transition of a WFN is associated with a color class or with a color domain.

5.1.2. WFN Color Functions

Color functions are formal sums of guarded functions built by standard operations (linear combination, composition, etc.) on basic functions.

There are three basic functions: *identity function* is a projection which selects an item of a tuple and is always denoted by a typed variable (e.g. X, Y) in application; *diffusion function* is a constant function which returns the bag composed by all the colors of a class or a subclass and is denoted All(C) where C is the corresponding (sub)class; *successor function* applies on an ordered class and returns the color following the given color, which is denoted as $\oplus$.

5.1.3. Guards

Color functions are formal sums of guarded functions built by standard operations (linear combination, composition, etc.) on basic functions.

An atomic predicate can identify two variables ([X = Y]), compare a variable with another using successor function ([$X = \oplus Y$]), or restrict a variable to be within a static subclass D ([X $\in$ D]).

The constrains above provide WFN with a good structure and simplify its analysis. The formalism of basic functions emphasizes the system symmetries. However, some asymmetric behaviors of objects in a given class are also supported by subclass divisions or by guards on transitions or on color functions, which has strengthened the modeling power of WFN.

5.2. WFN Modeling Tools

CPN-AMI (Kordon & Paviot-Adet 1999) allows users to build and analyze models of AMI-Nets, which are WFNs with a specific syntax.

GreatSPN (Chiola et al. 1995) is a friendly framework allowing the modeling, validation, and performance evaluation of Generalized Stochastic Petri Nets (GSPN) and their colored extension: Stochastic Well-Formed Nets (SWN). This tool also supports timed Petri Net based modeling and implements several efficient analysis algorithms to facilitate complex applications.

Besides these tools supporting WFN, one could also choose from a variety of tools for Colored Petri Nets and High-level Petri Nets to build their WFN models with respect to the WFN definition.

6. WFN MODELING PATTERNS FOR TRAIN CONTROL SYSTEM

In this section, we propose three modeling patterns that could be useful to build WFN models for railway control systems.

1. An equivalent structure in WFN to the arcs using IF-THEN-ELSE expressions defined in CPN Tool (Jensen et al. 2007);
2. The definition and implementation of a successor function;
3. A WFN queue structure with its corresponding management operations (adding item, removing item, modifying item, query, etc.).

We will define these modeling patterns with respect to a practical railway train control model. While these modeling patterns can also be applied to other complex system models.

6.1. IF-THEN-ELSE Arc in WFN

IF-THEN-ELSE is a common alternative structure that facilitates the modeling of some system logic functions. An arc using IF-THEN-ELSE expression is supported by some tools e.g. CPN Tools. Unfortunately, it is not supported in WFN. In this section, we propose two solutions to use IF-THEN-ELSE arc based on guarded functions and guarded transitions respectively.

6.1.1. IF-THEN-ELSE Arc by Guarded Functions

Consider a transition t and a place p. Let $\mathrm{C(t)} = \mathrm{C}$ and $\mathrm{C(p)} = C_1 \times C_2 \times \cdots \times C_k$.

We define F_t and F_f two unguarded colored functions, which are sums of tuple of basic functions.

$$F_t = \sum_m \langle f_1^t, f_2^t, \dots, f_k^t \rangle,$$
$$F_f = \sum_n \langle f_1^f, f_2^f, \dots, f_k^f \rangle.$$

We define a general IF-THEN-ELSE expression which labels an arc connecting a transition t and a place p:

$$\mathrm{W}^*(p,t) = if\ g\ then\ F_t\ else\ \mathrm{F_f}, \text{ where } * \in \{+,-\}.$$

As such an expression $W^*(p,t)$ is not supported by WFN syntax, we define the equivalent function $W_E^*(p,t)$:

$$\begin{aligned} W_E^*(p,t) &= [g]\, F_t + [\neg g]\, F_f \\ &= [g] \sum_m \langle f_1^t, f_2^t, \ldots, f_k^t \rangle + [\neg g] \sum_n \langle f_1^f, f_2^f, \ldots, f_k^f \rangle \\ &= \sum_m [g] \langle f_1^t, f_2^t, \ldots, f_k^t \rangle + \sum_n [\neg g] \langle f_1^f, f_2^f, \ldots, f_k^f \rangle \end{aligned}$$

where $* \in \{+,-\}$.

Obviously, $W_E^*(p,t)$ respects the definition of WFN standard functions (Chiola et al. 1991) and has the same semantic as the IF-THEN-ELSE expression $W^*(p,t)$.

6.1.2. IF-THEN-ELSE Arc by Guarded Transitions

Some Petri Nets tools do not support the concept of guarded function. In this case, we can use two guarded transitions to model the "then" and "else" clause of the IF-THEN-ELSE arc respectively.

Figure 4 shows an example of an IF-THEN-ELSE arc and its context. G is the guard of transition t (it is possible that G=TRUE) and g is the condition in the IF-THEN-ELSE expression. The other notations will be the same as defined in section 6.1.1.

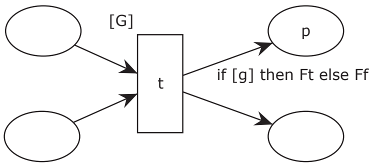

Figure 4 IF-THEN-ELSE Arc by Guarded Transitions

We propose an equivalent structure taking into consideration three cases based on the relationship between G and g.

Case 1: G is stronger than g ($G \subset g$)
In this case, $\{c \in C \mid G(c) = TRUE\} \subset \{c \in C \mid g(c) = TRUE\}$.
According to the firing principles, transition t is not enabled with a color $c \in \{c \in C \mid g(c) = FALSE\}$. Consequently, only the function F_t in the THEN-clause should be considered. Thus, the incidence function $W^*(p,t) = if\ g\ then\ F_t\ else\ F_f$ is then rewritten as $W^*(p,t) = F_t$, where $* \in \{+,-\}$.

Case 2: G and g are disjoint ($G \cap g = \emptyset$)
In this case, $\{c \in C \mid G(c) = TRUE\} \cap \{c \in C \mid g(c) = TRUE\} = \emptyset$
In opposition to Case1, transition t is not fireable with any color $c \in \{c \in C \mid g(c) = TRUE\}$, so only the function F_f in the ELSE-clause should be considered. Thus, the incidence function $W^*(p,t) = if\ g\ then\ F_t\ else\ F_f$ is then rewritten as $W^*(p,t) = F_f$, where $* \in \{+,-\}$.

Case 3: general case not belonging to Case 1 nor Case 2
In this case, we partition the colorset satisfying the guard G (i.e. $C_G = \{c \in C \mid G(c) = TRUE\}$) into two sub-colorsets C_G^T and C_G^T :
$C_G^T = \{c \in C \mid G(c) = TRUE \text{ and } g(c) = TRUE\}$,
$C_G^F = \{c \in C \mid G(c) = TRUE \text{ and } g(c) = FALSE\}$,
such that $C_G = C_G^T \cup C_G^F$ and $C_G^T \cap C_G^F = \emptyset$.
Then one models transition t with two transitions t^T and t^F defining $WN' = (P', T', Cl', C', W^{-\prime}, W^{+\prime}, \Phi', M_0')$, where

- $P' = P$; $Cl' = Cl$; $M_0' = M_0$;
- $T' = \{ T/\{t\} \} \cup \{t^T, t^F\}$;
- $C'_{t^T} = C'_{t^F} = C_t$;
- Assume $W^*(p,t) = if\ g\ then\ F_t\ else\ F_f$, $* \neq **$ and $*, ** \in \{+,-\}$, then $W^{*\prime}(p,t^T) = F_t$, $W^{*\prime}(p,t^F) = F_f$, $W^{**\prime}(p,t^T) = W^{**\prime}(p,t^F) = W^{**}(p,t)$;
- $\forall\, p' \neq p \in P'$, $* \in \{+,-\}$, $W^{*\prime}(p',t^T) = W^{*\prime}(p',t^F) = W^{*\prime}(p',t)$;
- $\forall\, t' \in \{ T'/\{t^T, t^F\} \}$, $\forall\, (p' \in P')$, $W^{+(-)\prime}(p',t') = W^{+(-)}(p',t')$;
- $\emptyset'(t^T) =$ G and g, $\emptyset'(t^F) =$ G and ($\neg$g).

We will give an example later in section 6.3 when introducing the operation 3 for a train queue structure.

6.2. Predecessor Function and Its WFN Realization

In WFNs the successor function $\oplus X_i^j$ is defined as an elementary function. While in some modeling cases it is also necessary to use a predecessor function, which is not defined in WFNs. This study proposes a method to use predecessor functions that will be noted as $\ominus X_i^j$. This study also gives its application constrains. With respect to these constrains one could always find an equivalent WFN structure which behaves as a predecessor function.

Let $\ominus X_i^j(c)$ be an application from $c \in C = C_1^{e_1} \times \cdots \times C_k^{e_k}$ to the predecessor of c_i^j in C_i, where C_i is an ordered class. It is worth noting that like the successor function, the predecessor of the first item in C_i is the last item.

To benefit from the features of WFN, when analyzing such a colored net using predecessor functions defined above, we could transform it to an equivalent WFN.

Figure 5 shows an example of a colored net using predecessor (Figure 5 (a)) and its equivalent WFN (Figure 5 (b)). In the example $C(t) = C \times C \times C$; X, Y are two identity functions that $X = X^1$, $Y = X^2$; (X-1) and (X+1) are notations of predecessor and successor functions that (X-1) = $\ominus X^1$, (X+1) = $\oplus X^1$.

Figure 5 (a) uses the predecessor function (X-1) in the output arc of transition t. In order to replace this structure using WFN, we do the following two steps:

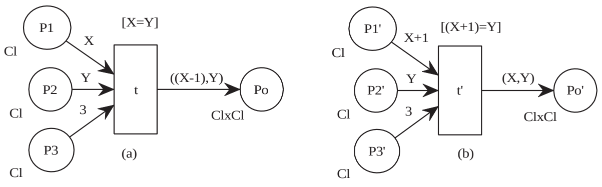

Figure 5 Predecessor and the Equivalent WFN

Step 1: search for all the instances of the identification function X in the "context of transition t", and replace them with the corresponding successor function (X+1).
It is worth noting that the three atomic predicates defined in WFN are replaced by the following rules respectively:

1. [X = Y] is replaced by [$Y = \oplus X$];
2. [$X = \oplus Y$] is replaced by [$\oplus X = \oplus Y$], which means [X = Y];
3. [X $\in$ D] is replaced by [$\oplus X \in$ D], which is not a WFN guard. In this case, let $D = \{x_m \cdots x_n\}$ be a subclass, we define a new subclass $D' = \{x_{m-1} \cdots x_{n-1}\}$ where x_{m-1} and x_{n-1} are the predecessors of x_m and x_n, respectively. Then [$\oplus X \in$ D] is transformed to [$X \in D'$].

In the example, the two instances are found in the guard of transition t and on the output arc from the transition t respectively in Figure 5 (a), which are then replaced by (X+1) in Figure 5 (b).

Step 2: replace the predecessor function (X-1) (in Figure 5 (a)) with the corresponding successor function X (in Figure 5 (b)). In the example, the one on the output arc of transition t is replaced by t'.

Application constrains: In order that the replacement above can be performed, for a color instance X_i^j, if the predecessor function $\ominus X_i^j$ is used, the corresponding successor function $\oplus X_i^j$ cannot appear in the "context of the same transition t", which includes the arcs connected with transition t and the guard of transition t. In other words, we cannot use the predecessor and the successor function of a same color instance X_i^j simultaneously and in the "context of a transition".

6.3. Queue Structure in WFN

While modeling railway control systems, more exactly the RBC model needs to have a centralized storage of the trains' queue, i.e., the information of all the trains in the railway line it manages. The information includes at least the trains' identifications, their positions and the sequence of these trains.
Using WFN, we can use a token of a product domain (e.g., $< TrainID \times Position >$) to illustrate the identification and position of each train. However, it is difficult to establish an ordered relation among these tokens.
In some software for modeling high level Petri nets such as CPN Tools (Ratzer et al. 2003; Jensen et al. 2007), it is possible to use a "list" type, like that defined in most programming languages, to realize this queue of trains. While the use of "list" type color class will lose the convenience of analyzing a WFN (a colored net using "list" type is obviously not a WFN).
This section defines a queue structure in WFN. It establishes an order relation among different elements and supports several operations e.g. insert, removal, query, and update. In addition, a colored net using this WFN-compatible queue structure remains a WFN, maintaining all its advantages for its analysis.

The proposition of this queue structure is faced with the requirements of modeling a practical train control system. Its application will be illustrated with the implementation of the Movement Authority (MA) function as part of the RBC model in Section 7. The implementation of the queue structure (e.g. Operation 3) uses the modeling patterns proposed in section 6.1 and 6.2.
Some basic declarations used in the queue structure are defined as follows:

CLASS	POS = <0> ∪ <1, 2, …, N> ∪ <N+1>;
	TID = <T(0), T(1), T(2), … , T(M)>;
DOMAIN	TRAINITEM = <POS, TID, POS>.

The color class POS is ordered and is divided into three sub-classes. Each position in <1, 2, …, N> represents a particular block in the railway line (which has N blocks; N is consequently a parameter that is bound to a specific value for each real line). The other two sub-classes <0> and <N+1> are for special purposes and will be explained in the following paragraphs. For convenience, we define a constant HEAD = N+1 for the following parts of this paper.
The color class TID enumerates the different identifiers of trains, in which T(0) is reserved as a special value and it does not represent a real train. TID could be an unordered class.
The color domain TRAINITEM is a 3-tuples Cartesian product and has the following practical meaning, as shown in Figure 6.

Class	Current Block	TrainID	PrevTrain's Block
Type	Position	TID	Position

Figure 6 Structure of TrainItem

Each token (except the token *TrainQueueRear*) of color domain TRAINITEM represents a particular train (*TrainID*) with its current position (*Current Block*). In addition, each train is connected to its previous train by indicating the block where its predecessor is located (*PrevTrain's block*). The following two special values help to construct the queue structure:

First Train: The first train in the railway line has not a preceding train regarding the actual state of the line. Let us give a special value "HEAD: POS" to its third field. As defined above, the constant HEAD = N+1. The block "N+1: POS" does not exist in the railway infrastructure. This value is used to indicate the first train in the queue.

Train Queue Rear: It doesn't present a real train, but offers a link to the rear train's position. The first and second fields of this item are always "0: POS" and "T0: TID", which is used to identify this rear item. It is worth noting that the block "0:POS" is not a real block, neither T(0) a real train. Its third field indicates the position of the rear train in the queue.

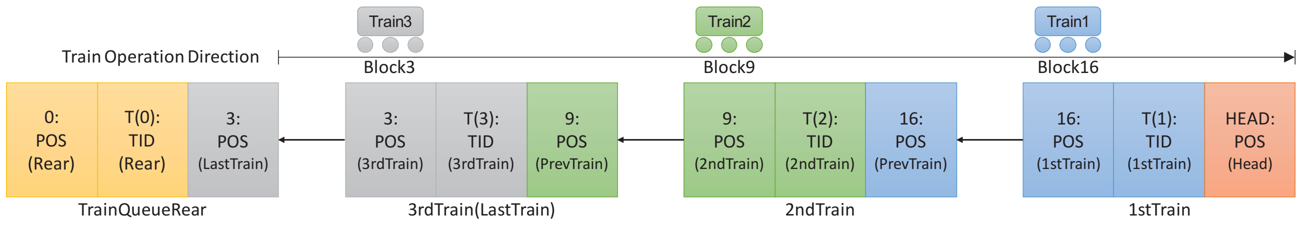

Figure 7 Conceptual List Structure of 3 Trains

The queue structure is then constructed by two places, i.e., place *TrainQueue: TRAINITEM* and place *FreeBlock: POS*.
In place *TrainQueue* there are tokens of color domain "TRAINITEM". In a special case where there are no trains in the railway line, the place *TrainQueue* is not empty, there still exists a token (*Train Queue Rear* in which the third value is "HEAD: POS"): <0: POS, T(0): TID, HEAD: POS>.
Tokens in Place *FreeBlock* represents the free blocks that are not occupied by a train. Each time a train moves, it will take the new position token from Place *FreeBlock* and release the token of its previous position.
To illustrate how to model a practical queue structure of trains in WFN, here is a general case assuming that there are 3 trains in the railway line, as shown in Figure 7.
Now it is necessary to define some basic operations to manage the queue structure.

Operation 1: Insert Operation
A new train is always inserted from the rear of the queue and it is normally inserted in Block 1. Then the objective of this operation is to insert a new token with <TrainID = tr: TID, CurrentBlock = 1: POS> to the queue and to modify the concerned tokens. This operation is explained with Figure 8, where *tr* is the identifier of the train to insert, and *p_last* is the position of the last train before this inserting operation.

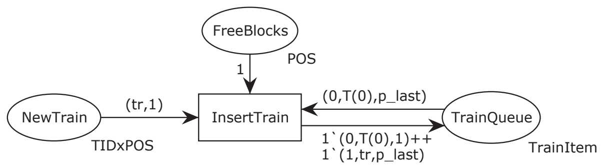

Figure 8 Insert Operation

For Operation 1, it is worth noting that:

- If there is already a train in block 1 before the operation, the token <1: POS> is no longer in place *FreePlace*, the new train to be inserted needs to wait until this block 1 is set free again;
- The operation also considers the case that the railway line is previously empty, i.e. p_last = HEAD: POS.

Operation 2: Removal Operation
When a train arrives at the end block (Block N) of the railway line and then leaves this railway line, its token <N:POS, tr:TID, HEAD:POS> must be removed from place *TrainQueue* and the token of the block <N: POS> must be released to place *FreeBlock*.
Figure 9 shows that the two tokens representing the first train <N, tr, HEAD> and its successor train <p1, t1, N> are involved. The last train token is removed and the "PrevTrain" field of train "*t1*" is updated to "HEAD: POS" as it becomes the first train in the railway line.

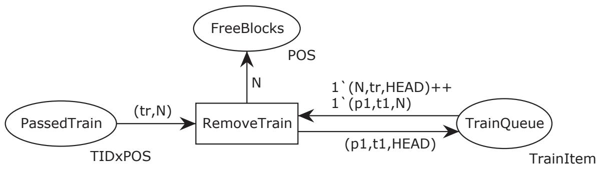

Figure 9 Removal Operation

For Operation 2, it is worth noting that:

- The variable "tr" is in fact, not necessary as it could be implied by only using its position "N", however we still use it to guarantee the right train we want to remove.
- When the train to be removed is the only train in the queue, its successor train is the rear item, i.e. "p1=0" and "t1=T(0)". In this case the removal operation will result in the case with a unique token <0: POS, T(0): TID, HEAD: POS >, which means there is no more trains in the railway line.

Operation 3: Request of Movement Authority (MA) for a train.
In order to avoid the collision of trains, each train must request the RBC for MA. By receiving the MA requested, the train knows to which block it can advance safely without any collision risk. In practice, it can advance until the anterior block to the current position of its predecessor train. This authorized position is called the End of Movement Authority (EOA). Normally the train needs to request a new MA regularly before reaching its EOA, in order not to be stopped during its advancement.

- When the considered train is the first train in the railway line, it can advance until the last block of this railway line, so its EOA is position N;
- When the considered train is not the first train on the railway line, and its predecessor train is currently in block "p_pre", then its EOA should be the block "p_pre - 1".

Therefore, the EOA position for a particular train with "tid=tr" could be expressed as "IF (p_pre = HEAD) THEN *N* ELSE (p_pre - 1)", which contains an "IF-

THEN-ELSE" arc and a predecessor function. Such an arc expression is not supported by WFN. While, after the application of equivalent structures proposed in section 6.1.2 and 6.2 (considering the definition HEAD = N+1), the structure of this operation using WFN can be given in Figure 10.

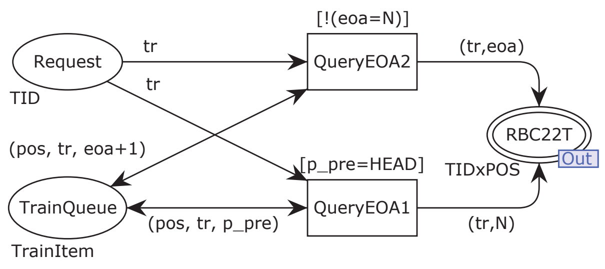

Figure 10 Request of Movement Authority

Operation 4: Update of Train Position
The update of train position does not affect the order relation of trains in the queue. Transition Update1 replaces the train's positon with a new value while transition Update2 deals with its successor train.
Figure 11 illustrates the WFN implementation of this operation. When the position value is updated, the previous position token "p0" is released to place FreeBlocks and the new position value is taken. We use two guarded functions with the guard [p<>p0] to avoid the manipulation to place FreeBlocks when the new value equals the old one.
The update operation is always triggered when RBC receives a position report < tr: TID, p: POS > from a train. After the update in its database, it is necessary to send back an acknowledgement to the train.

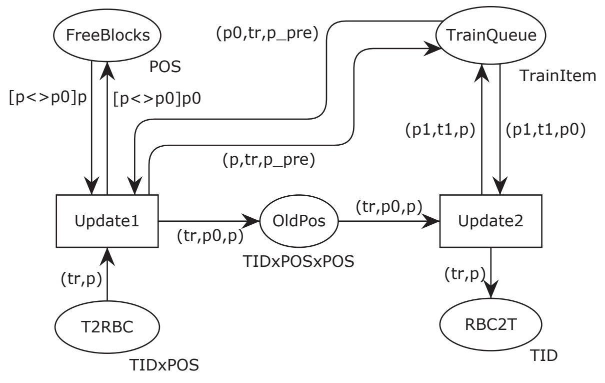

Figure 11 Update of Train Position

7. CASE STUDY

Faced with the practical problem of railway system controllers design, we have built several control models. Three models will be explained in this section. They offer the functions of managing multiple trains in a railway line, and with respect to WFN definitions.

7.1. System Structural Model

Figure 12 shows the model of the system architecture. The models are built in a modular way. The rectangles with double-line borders are modules.

This example model considers two train modules, whose detail is explained in section 7.2. A RBC module is built for the railway line management; and its details will be given in section 7.3.
Place Train2RBC and place RBC2Train represent the wireless interfaces between trains and the RBC module. The tokens in them are messages between different modules. In a similar way, place Balise2T models the Eurobalise interfaces. In our study, the Eurobalises are used to inform the trains of their locations.
The places T1info and T2info define the respective identifier of each train. Bidirectional arcs are used as the identifier tokens should never be consumed or modified. This can also be done for RBCs, in case of the modeling of a line controlled by several RBCs.
As all the train modules are exactly identical (except their TIDs as initial markings), it is possible to add more train modules to the architectural model, as long as these train modules are connected to the suitable interfaces and assigned with a TrainID.

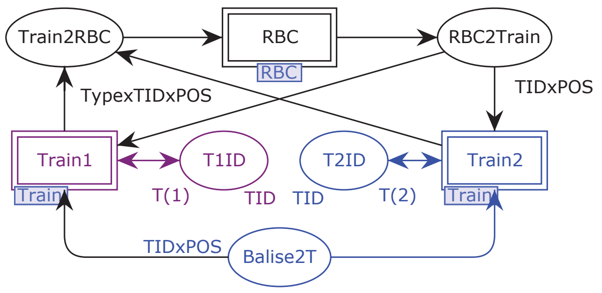

Figure 12 Structural Model of the Case Study

7.2. Train Model

Figure 13 gives the train model integrated with the functions to enter a railway line, to advance with respect to its MA, and finally to pass this railway line.

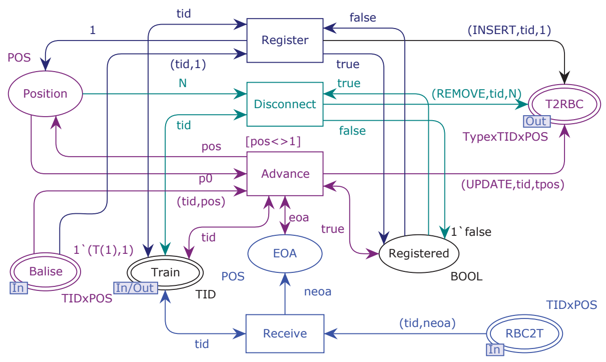

Figure 13 Train Module of Case Study

Let us suppose that initially the train just arrives on the first block (place Balise: 1, place Position is empty, place EOA: 1, place Registered: false). From this initial state, the following functions describes the behavior of the train sequentially.

Register function is to register the train itself to the RBC. To fire the transition Register, the train must be located in block 1 (token "1: POS" in place Balise) and the state

Registered is false. When transition Register fires, it will send a message of type "INSERT" to the RBC, put a token "1: POS" into place Position, and change the state Registered to true.

Advance function is to simulate the advancement of a train already in the blocks. The train can advance as well as it does not arrive on its EOA position. Each time the train passes a block, its new position is received via place Balise so that transition Advance is fired. The token in place Position is updated, and a position report is also sent to inform the RBC of its new position.

Transition Receive can be fired when there is a MA message generated by RBC. Place RBC2T are shared by all the trains, so only the message for this train (tid) will be received. After receiving the message, its new EOA value is then memorized in place EOA.

Disconnect function is to inform the RBC that it has passed the railway line. After passing the last block (block N), the transition Disconnect can be fired. Then the token "N: POS" is removed from place Position, the train sends a message of type "REMOVE" to the RBC and changes its state Registered to false.

7.3. RBC Model

Figure 14 represents the RBC model. The four main functions (e.g., InsertTrain, RemoveTrain, QueryEOA and PositionUpdate) are well explained as the four operations of train queue structure in Section 6.3.
What we need to add in this model is the way to fire different functions. The functions InsertTrain, RemoveTrain and PositionUpdate can be fired after receiving a message from a train. A field "Type" (i.e., INSERT, REMOVE or UPDATE) in the message helps to choose the corresponding functions to fire.
For convenience, the RBC model regards a position report as a MA request. So, each time it receives a position report, the trainID is then put into place Request in order to generate a MA for it. The RBC also generates a MA for a train that is just registered (after transition InsertTrain is fired).

8. CONCLUSIONS AND PERSPECTIVES

In this paper, we have shown that it is possible to use WFN to model complex systems such as railway systems by using several modeling patterns and techniques that we propose. These modeling patterns also make it possible to model some structures and extensions of other types of colored Petri nets such as the CPNs defined in (Jensen, 81), based on the WFN rules (e.g., elementary colored functions). We illustrate our propositions by applying them to the Movement Authority (MA) function modeled in the ECTS-2 context.
The prospects for this work are of course to continue the modeling of other functions of a railway system with a view to its complete automation. We are thinking in particular about the routing function of a train inside a node, which is implemented today in a semi-automatic mode, which requires a man-machine cooperation.
Beyond the modeling stage, it will be necessary to complete this work by the development of a method allowing the formal verification of our models while controlling their combinatory explosion. On the other hand, we want to use the techniques of reductions applicable to the WFN for our developed models. We also plan to directly use the calculation of colored invariants on reduced models but also the construction if necessary of symbolic reachability graph. All these mentioned methods may be complemented by the proposal of a formal model verification such as the assume-guarantee reasoning (Nguyen Huu 2013) in order to ensure that the global model inherits the properties verified on its component modules.

ACKNOWLEDGMENTS

This study is carried out within the framework of the CompRAIL project of ELSAT2020. The ELSAT2020 research program is co-financed by the European Union with the European Regional Development Fund, the French state and the Hauts de France Region Council.

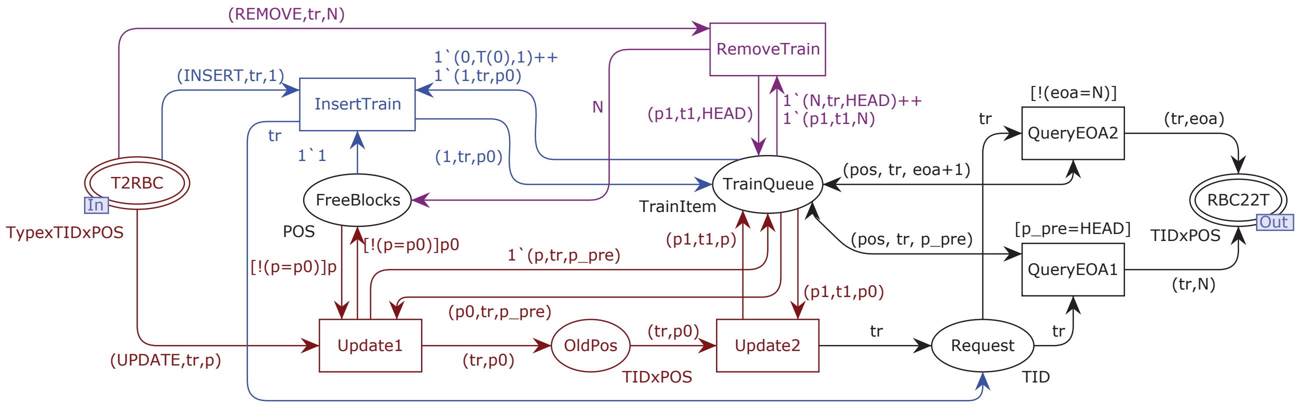

Figure 14 RBC Module of Case Study

REFERENCES

Evans W.A., 1994. Approaches to intelligent information retrieval. Information Processing and Management, 7 (2), 147–168.

van der Aalst, W.M.P., Stahl, C. & Westergaard, M., 2013. Strategies for Modeling Complex Processes Using Colored Petri Nets. In K. Jensen et al., eds. Transactions on Petri Nets and Other Models of Concurrency VII. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 6–55.

Berthelot, G. & Lri-Iie, 1986. Checking properties of nets using transformations. In G. Rozenberg, ed. Advances in Petri Nets 1985. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 19–40.

Chiola, G. et al., 1995. GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. , 24, pp.47–68.

Chiola, G. et al., 1991. On Well-Formed Coloured Nets and Their Symbolic Reachability Graph. In High-level Petri Nets SE - 13. Springer, pp. 373–396.

Couvreur, J.M. & Martínez, J., 1991. Linear invariants in commutative high level nets. In G. Rozenberg, ed. Advances in Petri Nets 1990. Springer Berlin Heidelberg, pp. 146–164.

Diaz, M., 2013. Petri Nets: Fundamental Models, Verification and Applications M. Diaz, ed., London, UK: John Wiley & Sons.

Esparza, J. & Hoffmann, P., 2016. Reduction Rules for Colored Workflow Nets. In Fundamental Approaches to Software Engineering: 19th International Conference (FASE 2016). pp. 342–358.

Haddad, S., 1991. A Reduction Theory for Coloured Nets. In K. Jensen & G. Rozenberg, eds. High-level Petri Nets: Theory and Application. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 399–425.

Janczura, C.W., 1999. Modelling and Analysis of Railway Network Control Logic using Coloured Petri Nets. University of South Australia.

Jansen, L., Meyer zu Hörste, M. & Schnieder, E., 1998. Technical Issues in Modelling the European Train Control System (ETCS) Using Coloured Petri Nets and the Design/CPN Tools. In K. Jensen, ed. Workshop on Practical Use of Coloured Petri Nets and Design. Daimi PB-532, Aarhus, Denmark: Aarhus University, pp. 103–115.

Jensen, K., 1981. Coloured Petri Nets and the Invariant-Method. Theoretical Computer Science, 14(3), pp.317–336.

Jensen, K., Kristensen, L.M. & Wells, L., 2007. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer, 9(3–4), pp.213–254.

Kordon, F. & Paviot-Adet, E., 1999. Using CPN-AMI to Validate a Safe Channel Protocol. In Proceedings of the International Conference on Theory and Applications of Petri Nets - Tool presentation part. Williamsburg, USA.

Meyer zu Hörste, M., 1999. Modelling and Simulation of Train Control Systems Using Petri Nets. In FMRail Workshop.

Murata, T., 1989. Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 77(4), pp.541–580.

Nguyen Huu, V., 2013. Modular Verification of Petri nets. UNIVERSITY OF BORDEAUX 1.

Ratzer, A.V. et al., 2003. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In Proceedings of the 24th international conference on Applications and theory of Petri nets (ICATPN'03). pp. 450–462.

Vanit-Anunchai, S., 2014. Experience using Coloured Petri Nets to Model Railway Interlocking Tables. In 2nd French Singaporean Workshop on Formal Methods and Applications (FSFMA'2014). Singapore, pp. 17–28.

Vanit-Anunchai, S., 2010. Modelling Railway Interlocking Tables Using Coloured Petri Nets. In D. Clarke & G. Agha, eds. Coordination Models and Languages: 12th International Conference (COORDINATION'2010). Amsterdam, The Netherlands: Springer Berlin Heidelberg, pp. 137–151.

Vanit-Anunchai, S., 2009. Verification of Railway Interlocking Tables using Coloured. In The 10th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. DAIMI PB 590, Department of Computer Science, University of Aarhus, pp. 139–158.

Xie, Y., Khlif-bouassida, M. & Toguyeni, A., 2016. Modeling Of Automatic Train Operation Control Using Colored Petri Nets. In 11th International Conference on Modeling, Optimization & Simulation (MOSIM 2016). Montréal, Canada.

Žarnay, M., 2004. Use of Petri Net for Modelling of Traffic in Railway Stations. In Proceedings of international conference Infotrans. Pardubice.