DOI: https://doi.org/10.46354/i3m.2019.imaaca.014

COMPONENT APPROACH BASED ON PETRI NETS FOR THE DESIGN OF THE AUTOMATIC CONTROL FOR RAILWAY SYSTEM

Armand Toguyéni

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

armand.toguyeni@centralelille.fr

ABSTRACT
The automation of rail systems is a major challenge for the development of this mode of transport. This automation must affect all the functions of the control system and not just the replacement of train drivers. This study proposes a component approach for modelling control functions based on Colored Petri Nets. This component approach masks the complexity of the system components and their functions from the designers of a rail system. In this work we also propose a new formal verification method based on the construction of a reduced reachability graph of a global model. This approach makes it possible to verify the main properties of the components necessary for their implementation in software libraries that can be used by railway system designers.

Keywords: Discrete Event Systems, Colored Petri Net, formal modelling and verification, Automatic Train Control.

1. INTRODUCTION

The automation of rail systems is a major challenge for the development of this mode of transport with regard its competition with road and air transport. This study proposes to use a component approach to facilitate the design of railway systems. This requires to develop generic component libraries. Assisted by such libraries, a designer can model a rail system by instantiating the generic components of his library and specifying the interactions between these components. This requires to check that each generic component works properly.

This work proposes to use Jensen's Colored Petri Nets (CPN hereafter) for modeling railway systems. They allow to use the modularity and the parametric modeling to build generic components. These generic components can be instantiated to build a global model.
This paper is structured as follows. In the second section, we will present how actual railway systems operate and the main functions of an automatic control system. In section 3, we will propose a decentralized control architecture for the implementation of automatic control of railway systems. In the fourth section, we will present our modeling principles based on the concept of Petri Nets modules, with modules whose operation depends on parameters. In the fifth section we will propose a modeling of some of the components presented in the third section. The sixth part focus on the verification of our generic components. In particular, we will introduce a new semantics of PN interpretation in order to reduce the size of reachability graphs. Section 7 gives a case study to illustrate our approach. We will end with a conclusion and perspectives.

2. STATE OF ART OF AUTOMATIC TRAIN CONTROL

2.1. Description of a railway system and its operation in traditional mode

A railway system can be abstracted as consisting of railway nodes and railway lines. There are two categories of railway nodes: stations and junctions. A rail junction consists of switches end track elements that establish routes for routing trains. Stations have platforms that allow a train to stop for passenger loading/unloading. This is what differentiates them from junctions (Lusby, Larsen, Ehrgott, & Ryan, 2013). Rail lines are used to connect stations together. But several railway lines may cross at junctions allowing trains to move from one line to another to reach a destination station. Lines generally have two one-way tracks to connect two nodes in both directions of traffic (round trip). Normally, all trains on a track move in the same direction of travel. Some lines may have bidirectional track portions. Within a railway node, some track sections can also be bidirectional.

Safety is one of the main criteria for the proper functioning of a railway system. To do this, it is necessary to avoid collisions.
In order to ensure the proper functioning of railway systems, three categories of human operators take decisions and manually trigger control operations: train drivers, line regulators and railway dispatchers.

The role of the train driver is to control the advance of his train by respecting the signaling (lights and speed limits). He has no control over the train's itinerary, which depends on the other two operators. Today, in modern control and signaling systems, the driver operates under the control of an automated train protection system (ATP). This system is used to guarantee safety, especially on the lines. It can trigger an emergency stop of the train if necessary. The driver performs other functions such as opening and closing train doors.
The function of the line regulator is to regulate the traffic of the trains on the line it controls. To do this, it can switch a slow train onto a bypass track to allow a faster train to pass it. The slow train then returns to the main track as soon as possible.
The dispatcher role is to decide which trains pass through the node he controls. It establishes a route for each train. It can be assisted by a computer for assigning a route to a train. By default the signal at the node input is closed (red light). When the signal is open (green light), the train can enter the node (Lusby, Larsen, Ehrgott, & Ryan, 2013).

2.2. Automatic train control systems

Current rail systems are not very automated but there are many automated metro lines in the world because they are simpler. The main automatic control system (ATC) used by metros are Communication-Based Train Control also called CBCT (IEEE Std 14741, 2004). One of the main features of this system is the use of radio communication between trains and ground infrastructure. This system has inspired ERTMS Level 2 which is based on the GSMR communication system (European Railway Agency, 2016). This study assumes the use of such type of communication.
An automatic train control system is actually divided into three subsystems (Yin, et al., 2017): the train operating system (ATO), the automatic train protection system (ATP) and the train supervision system (ATS).

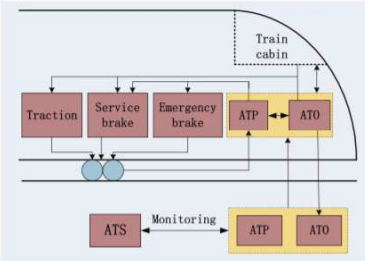

Figure 1: ATC structure (Yin, et al., 2017)

The function of the ATO is to replace the driver on board trains. This is the function to be developed as part of the autonomous train. It is responsible for controlling the advance (traction) and stopping of trains (braking) according to its operating modes and speed limits. It is a function that is both on board and on the ground (Figure 1). Indeed, the trackside controllers will calculate the train's movement authorizations according to the limitations due to its location in the network but also according to the position of the other trains in the network.
The role of the ATP function is to monitor the execution of train operations. In manual driving mode, it controls the commands given by the driver to the train. In automatic mode it controls the orders of the ATO. The ATP will directly control the emergency braking. It is also a function that is both on board the train and on the ground (Figure 1). The majority of current rail control and signaling systems are based on the blocks' technique. This technique guarantees safety on the tracks of a line. Indeed, each track is divided into electrically isolated blocks. Safety requires that there be only one train per block. In actual systems, the safety implemented by ATP is based on the concept of interlocking. Also, there are many studies on the verification of interlocking by formal methods including CPN (Vanit-Anunchai, S., 2014). ATP is a critical function from safety point of view.
The ATS function is a function of the ground system (Figure 1). It monitors that the train movements are in accordance with the planned scheduling. It is also responsible for the dynamic routing of the train for crossing railway nodes.

2.3. Conclusion

More specifically, this work is a contribution to the implementation of sub-functions of the ATO (train movement authorization) and ATS systems. For this, we will be inspired by ERTMS/ETCS level 2 which is in fact an ATP function for protecting the movement of a train on a railway track (European Railway Agency, 2016). In next sections, we propose a new architecture and a methodology to develop automatic control system.

3. DECENTRALIZED ARCHITECTURE FOR THE CONTROL OF A RAILWAY SYSTEM

This study concerns the management of multiple trains in a railway network with a full automation of the system. To propose a new architecture, we are going first to propose a structural and a functional decomposition of a railway network. A top-down approach is used to conduct this decomposition for a railway system by first considering separately the two points of view and then by mapping together the elementary components of the system with their functions.

3.1. Structural decomposition

A railway network is a complex system. Its control is distributed in several components of the ground infrastructure and onboard the trains. In order to be able to model such a system, a structural decomposition is necessary. Figure 2 summarizes the proposed decomposition.
It is important to notice that the structure of each subsystem (stations, junctions and railway tracks) is different depending on the railway network requirements, but the basic components (block, switch, track section, balise and track circuit) used to implement them are generic.

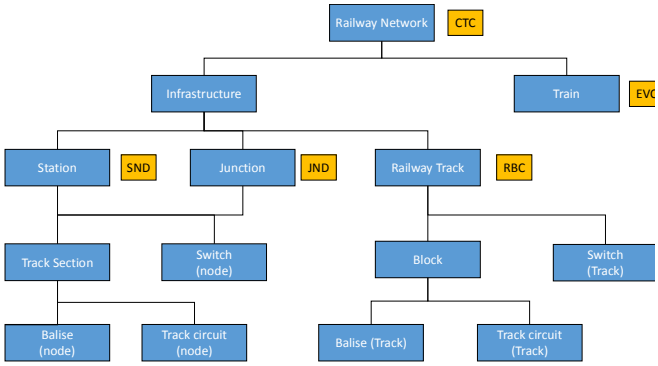

Figure 2: Structuration Decomposition of a Railway Network

3.2. Functional Decomposition

Because this study aims to propose a methodology for automatic control, the structural decomposition also shows the different controls centers that allow implementing a decentralized control of a railway system.

At the top of the hierarchy, there is the Centralized Traffic Control (CTC hereafter). The CTC checks in real time that the planned transport plan is implemented throughout the network. It has a global but macroscopic view of rail traffic in the system it controls. Thus, it defines and updates the itinerary of each train. Each itinerary is defined in terms of departure station, arrival station (final destination), intermediate nodes through which the train will pass. The arrival dates of a train at each node of its itinerary are defined by the planned transport plan. The CTC regularly receives feedback from the lower-level control centers that execute locally this plan. It calculates the differences between the executed and the planned plan and sends back to local centers, updated local transport plans. It implements rail traffic supervision that is an ATS function.

The types and number of local controllers are consistent with the breakdown of the infrastructure into railway nodes and lines composed of tracks. A local control center is associated with each railway node. Each junction is controlled by the JTC (Junction Traffic Controller) whose main function is to implement automatic train routing (an ATS function) within the node. Taking into account the local planned transport plan (list of trains to cross the junction in a time slot), the JTC allocates in real time the resources necessary for each train arriving at the node to set its route. It is based on a planned scheduling of traffic to cross the junction in accordance with the local transport plan received from the CTC. In case of fault, the local transport plan is updated. If this update does not absorb all the disruptions, the CTC is informed so that more global actions can be taken to find a solution. Trains can be slowed down, accelerated or even their itinerary modified. The JTC is also responsible to set up the train route (it is an ATO function) and to give to each train its movement authorization inside the junction (it is an ATP function).

The stations are controlled by the STCs (Station Traffic Controller). STCs implement automatic train routing like JTCs. In addition, they must manage the assignment of platforms to trains that stop at the station.

The Radio Bloc Center (RBC) operates in the same way as in ERTMS/ETCS level 2 (European Railway Agency, 2016). Each train regularly sends its position to it. Taking into account the position of each train on the track and their time constraints, the RBC calculates their respective movement authorizations (authorized travel distance and speed profile) and regulates the traffic of the trains on the line. Movement authorizations are transmitted to trains in response to their requests for movement authorization.

The fourth local control center is onboard in each train. It is implemented by the main computer of the train called EVC (European Vital Computer). It has a role similar to that defined in the standard (European Railway Agency, 2016). It implements both the ATP and ATO functions of the train. The role of this computer is to control the train's advance (an ATO function) by respecting the movement authorizations (an ATP function) transmitted by the RBC when it is on a track. When it must cross a node, it communicates with the controller of the node (JTC or STC) who gives it his movement authorizations in accordance with the route it has assigned.

3.3. Mapping Relations

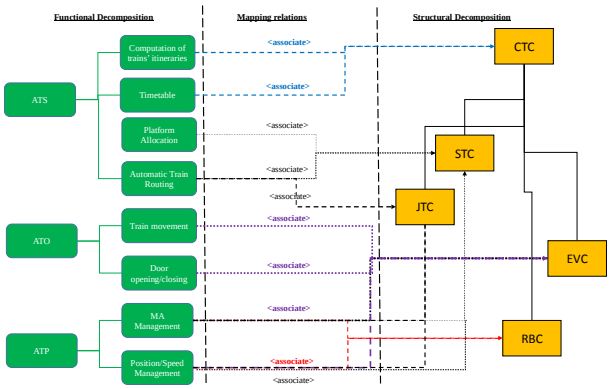

Figure 3: Mapping Relations from Control Viewpoint

Figure 3 depicts the relations mapping between the control components of a railway network and the different functions hold by each of them. On the left of the figure, the green nodes represent the functional decomposition of a Railway Network from control viewpoint. On the right of Figure 3, the yellow square model the main control components. As an example, one can see that STC is associate with platform allocation and automatic train routing that are ATS sub-functions and is also associate with MA Management and Position/Speed Management that are ATP sub-functions.

In the rest of the paper, we will focus on the design of control functions of JTC, RBC and EVC components.

4. MODELING OF DISCRETE CONTROL COMPONENTS

The objective of this part is to show how to model components in order to allow the modelling of a rail

ISBN 978-88-85741-31-7; Bruzzone, Dauphin-Tanguy and Junco Eds

control system. We focus here on the train EVC, RBC for line control and STC for junction control.

4.1. Colored Petri Nets

4.1.1. Introduction

For DES, there are three types of formal models (Cassandras and Lafortune, 2008): regular languages, automata and Petri Nets (PN hereafter). PN were defined in 1962 in the thesis of the German mathematician Carl Adam Petri. He then showed that it was the best formalism to model DES characterized by several subsystems evolving in parallel and sometimes having to be synchronized. The initial formalism is called, Places/Transitions Petri nets (notation P/T-nets). But since that time, many abbreviations or extensions of PN have been proposed by other authors. Colored Petri Nets (CP-nets hereafter) are basically an abbreviation for P/T-nets. It means that any colored model can be unfolded to find the equivalent P/T-nets model. Coloration consists of defining sets of objects. Thus, the model's tokens can model real-world objects. With P/T-nets, it is necessary to make a specific net for each object. CP-nets make it possible to factorize the behavior common to several objects into a single model of the same size as the model of an object in the case of P/T-nets. Thus, they make it possible to gain in concision and thus to reduce the size of the model representing the set of physical objects.

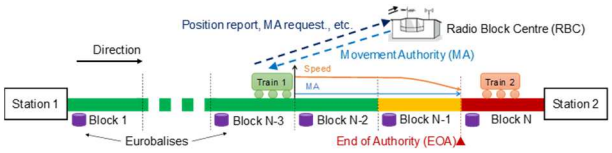

Figure 4: Example of two trains on a railway track

To illustrate the power of expression of Colored Petri Nets, consider the example given in Figure 4. This example defines two trains moving on a railway track composed of N blocks. N is a parameter that represents the length of the track in number of blocks. Figure 5 illustrates the modelling of this system in P/T-nets. In this case, in order to distinguish between the two trains, a model must be constructed for each train. The overall model of the system is therefore composed of 2N places and 2N transitions. The corresponding automaton would be composed of N^2 states illustrating the combinatorial explosion of this formalism.

Figure 6 illustrates the same problem modeled in CPN which is an extension of CP-Nets (Jensen et al., 2007). In this case, the problem parameters can be specified by constants (NTr for the number of trains and N for the number of blocks).

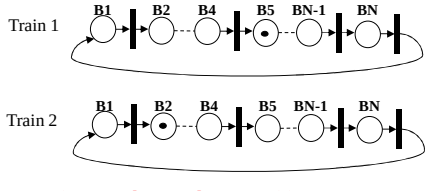

Figure 5: P/T-nets of the two trains on a track

This allows to define train identifiers (*Tr(1)* is the train identifier of train 1 and *Tr(2)* is the identifier of train 2) using the *ML language* of CPN Tools. Similarly, we can specify the block identifiers on which the trains are positioned *(B(2)* and *B(5)* respectively). CPN Tools' ML language allows to define composite types such as *OccBlock* which specifies the blocks occupied by a train. The definition of these types allows to fold the P/T-nets of Figure 5 and to obtain the CP-nets of Figure 6 composed of 1 place and 1 transition. It can be seen that this model is much more compact than the one obtained with P/T-nets or finite state automaton. However, it requires the creation of functions such as function *f*, which reflects the advance of each train on the rail track. But there are several types of CP-nets. In particular, this study is based on Karl Jensen's Colored Petri Nets (CPN hereafter (Jensen et al., 2007). They are not pure CP-nets. Actually, they become a High Level Petri Nets (Jensen and Rozenberg, 2012).

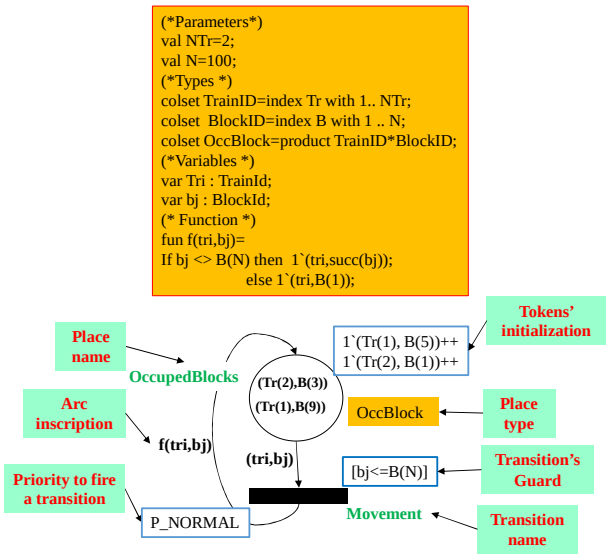

Figure 6: CP-nets of the two trains on a track

In CPN model, each place has a type such as *OccBlock* that defines the type of the place *OccupedBlocks*. It is initialized with two tokens that are each a 2-uple. Based on the operator of multiset ('++' is the addition operator for a multiset), if *M* represents the marking function, one can write the marking of *OccupedBlocks* by the equation (1).

$$M(\text{OccupedBlocks}) = 1`(\text{Tr}(1), \text{B}(5)\ ) + +1`(\text{Tr}(2), \text{B}(2)) \quad (1)$$

$1`(\text{Tr}(1), \text{B}(5)\ )$ is a token composed of 2 elementary colors: *Tr(1)* is the identifier of train 1 and *B(5)* defines the block that it occupies.

4.1.2. Main features

The choice of CPN is also justified by the existence of CPN Tools, a suite of tools for editing and analyzing them. It makes it possible to develop a model in a modular and hierarchical way (see section 4.3). CPN

Tools also offers designers the ability to analyze the properties of their models. The analyses are based on the generation of the reachability graph of the CPN model called occurrence graph by the authors (Jensen and Rozenberg, 2012). Its generation makes it possible to obtain a report on the usual properties of a PN such as boundedness or liveness. This feature will be used in section 6. CPN Tools has also a model-checker called ASK-CTL that allows to analyze specific properties of a model (Christensen and Mortensen 1996). It is out the scope of this study.

4.2. Component modelling principles

The objective of our modeling is to build generic components that can be placed in a library after checking their properties. These components can then be used to build global models of a railway system by instantiating these generic components. For that, each component is defined as a CPN module interface places to allow it communicating with its environment. Because the global model is distributed in different computers, the components communicate by the semaphore technique (Murata, 1989). The orientation of the arc linking an interface place to a module indicates the nature of the semaphore. So when a module (component) requests a service from another component, it will use a semaphore request. The requesting component then acts as a client and the receiving component as a server component. When the server component responds to the client component, it will use an acknowledge type semaphore.

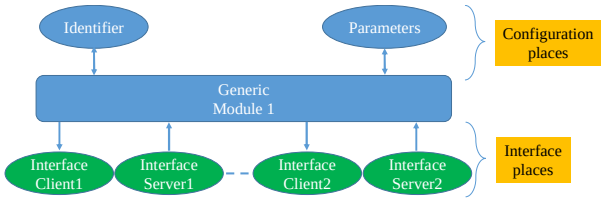

Figure 7: Generic component architecture

In order to be able "to instantiate" the generic components to implement a global model, let us use configuration places of each module. A configuration place is used to specify the identifier of each component at the time of instantiation (copy in the global model) of each generic module. One or more other places will make it possible to define the specific parameters of each component (Figure 7). It can be noted that the arcs linking the configuration places to the module are bidirectional. This means that the tokens of these places are read by the module and then rewritten in the place for later use.

Figure 8 shows how to build a global model from generic components. This global model uses two instances of module 1 and one instance of module 2. The configuration places are initialized to define the identifier and parameters of each module when building the system model. It is noted that the identifier of each instance is unique even if they are instances of the same module. The parameters of two instances of the same module can be identical. Thus, P1.1 and P1.2 can contain the same values. It is noted in this construction that the communication interfaces between two types of modules are merged. Thus it is the same place M1toM2 that models the communications of the instances of module 1 to the instances of module 2. This choice is made to illustrate the fact that in modern architectures communications are based on media operating in broadcast mode. Thus, the exchanged messages must contain the sender's identifier and the recipient's identifier so that the latter can recognize the messages sent to him and can respond to the sender.

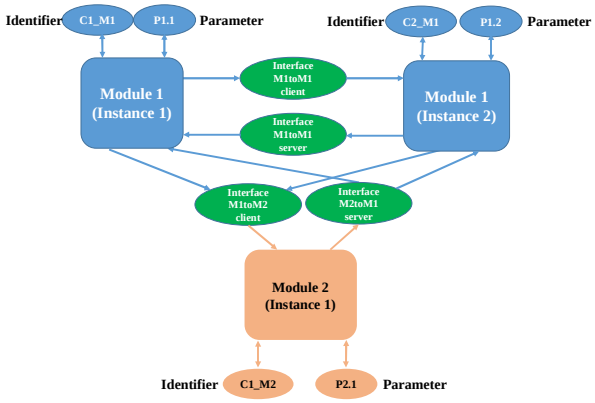

Figure 8: System architecture built from generic component instances

4.3. Implementation of modelling principles in CPN Tools

To implement our modeling principles, CPN Tools offers 3 specific services: the ability to define component identifiers based on index color set ('*colset*' type constructor in CPN's syntax), module definition through substitution transitions, and places' integration through socket/port or fusion places (Jensen and Rozenberg, 2012; Jensen and al., 2007).

An *index colorset* are sequences of values defined by an *identifier-name* and an *index specifier*. Other parameters are represented by their convenient types and can be merged into a *record* or *product* colset to have a compact representation. Figure 9 shows an example of the declaration of the parameters of a train component (EVC). Among these parameters, colorset *TRAINNO* is defined as an *index colset* and is a type used to define different identifiers of trains. The examples of its values could be *Train(1), Train(2)… Train(10)*. A *record colset TrainAttribute* assigns other parameters to an instance of the train component, such as train type, train mass, its origin and destination stations.

```
1 val maxtrain=10;
2 colset TRAINNO = index Train with 0..maxtrain;
3 var tno: TRAINNO;
4 colset TrainType = with Passenger | freight;
5 colset TrainMass = int with 0..100000;
6 colset StationName = string;
7 colset TrainAttribute = record tType:TrainType * tMass:TrainMass * tOrigin:StationName * tDesti:StationName;
```

Figure 9: Declarations for configuration places of component train (part of)

ISBN 978-88-85741-31-7; Bruzzone, Dauphin-Tanguy and Junco Eds

The notion of substitution transitions makes it possible to distinguish two modeling layers of a system under CPN Tools: the global layer (upper level) and the component layer (lower level). The *global layer* mainly defines the configuration of the instances of the components and illustrates the connections between them, as shown in Figure 11. This global layer is also modeled as a colored Petri net model. In CPN Tools, the module body of each instance is represented by a substitution transition (rectangles with double-line borders in Figure 10) in this global layer.

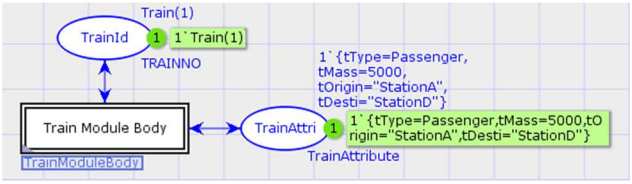

Figure 10: Example of configuration places of train component

The hierarchical feature supported in CPN Tools offers a possibility of implementing the interfaces between different component models or their instances. Figure 11 shows an example of system modeled with two layers: the *global layer* and the *component layer*. Two components "Train" and "RBC" are modeled using the component modeling method of the parametric module representation. In the example, one instance of each component is used to build a global model. Transitions "*Train*" and "*RBC*" are *substitution transitions* in CPN Tools and their details are modeled in the module body of the corresponding component modules. Places "*Train1*" and "*RBC1*" are configuration places used to assign different identifiers and parameters to these instances the corresponding generic components. Place "*T2RBC*" is an interface place which is used send information from a train instance to an RBC instance.

An interface modeled by CPN hierarchy is implemented by *port/socket assignments*, which are used to merge places on the two layers. Such a place on the *lower layer* (component layer) is called a *port*, and that on the *higher layer* (global layer) is called a *socket*. A *port* is always associated with a *port-type tag* (the blue tags in Figure 11) and can be one of the three kinds according to the direction: tag "In" for "*input*"; tag "Out" for "*output*" and tag "In/Out" for both the two directions.

A *socket* is an input place or an output place of a substitution transition, i.e. there is always at least one arc between a substitution transition and a socket.

By using the port/socket assignments, a component module can be "glued together" with the surroundings of its corresponding substitution transition. Each socket must be assigned to a port on the corresponding subpage. A port with a tag "In" must be assigned to a socket which is an input place of the substitution transition. Analogously, an "Out" port requires a socket which is used an output place of the substitution transition. In the example, the "In/Out" ports are used by the configuration places because that the identifiers and parameters in these places are normally to be referred to, other than to be generated nor to be consumed.

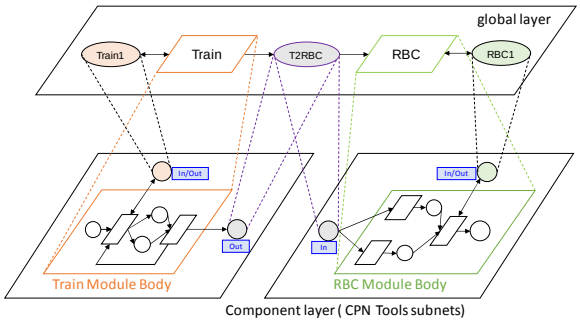

Figure 11: Modelling of interfaces by CPN Tools hierarchy

In order to distinguish the different instances of the same component as a concrete sender or receiver, a colored token to be used in the interface places need to be defined as a *product colset* as given by equation (2):

$$(\text{Sender Identifier}, \text{Receiver Identifier}, \text{Message}) \quad (2)$$

The "*Message*" in (2) can also be a *product colse*t, which is usually composed of a "*MessageType*" filed and one or more value(s). An example of a position report sent by a train instance to an RBC instance to update its position could be:

$$(\text{Train}\ (1), \text{RBC}\ (1), (\text{UPDATE}, \text{Train}(1), 10)) \quad (3)$$

In this message, "*UPDATE*" is the message type, followed by the values "*Train (1), 10*" that means *Train(1)* is located on block *10*.

5. MODELLING OF GENERIC COMPONENTS

In this section, we will propose models of the three generic components used by our actual global model of a railway system: EVC, RBC and JTC.

5.1. Modelling of the EVC component

As indicated in paragraph 3.2.3, the main functions of a train are the management of its position, the management of its movement authorizations and the management of its movement. The train's itinerary in the rail system is defined by the data of its departure and destination stations. Also, in its departure station, the train waits to receive a departure authorization from the STC. Following this authorization, it requests the STC for a route to go out of the station. The established route is notified to it by the STC. The train will then travel through the track sections to reach the first block of line at the station exit. In parallel, the station has pre-registered the train with the RBC managing the line the train will use (see the modeling of JTC in section 0 and the case study in section 7). A train departure operation is modeled by the CPN of Figure 12. The red part of the figure models the crossing of the last section of track before reaching the railway line. When the train enters the first block of the railway line, this is confirmed by the

ISBN 978-88-85741-31-7; Bruzzone, Dauphin-Tanguy and Junco Eds

signal sent by the block's balise (see place *Balise2EVC* in the upper left part of Figure 12). It then sends his position to the RBC to confirm his arrival on the line (see place *EVC2RBC* in in the upper right part of Figure 12). From that moment, it is registered by the RBC, which then sends it its first movement authorization (MA). This MA allows him to retrieve the last block number (End of Authorization or EOA) of the line it is authorized to join given the occupation of the line by other trains. The place *Balise2EVC* models the train's communication interface with its environment and in particular the reception of notifications of the train's position by the balises of the infrastructure. These notifications distinguish between line balise messages that are processed by the blue part and node balise messages that are processed by the purple part. The green part models the treatment of MA by ECV. It is noted that for reasons of simplicity, this model does not integrate the train's operating modes.

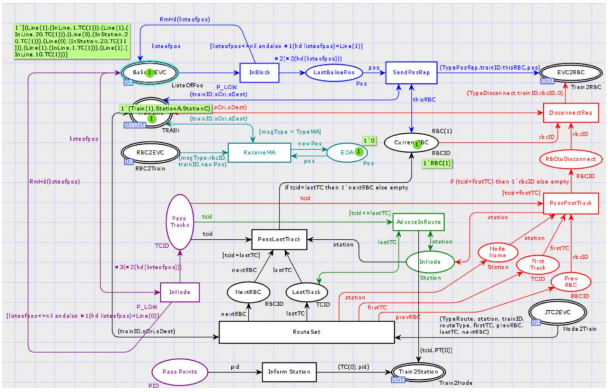

Figure 12: Model of EVC

5.2. Modeling of the RBC component

An RBC manages a line between two nodes (see Figure 4). Its main function is to regulate train traffic by giving them movement authorizations according to the location of each train on the line. Each train arriving on the line must be registered with the RBC for it to take into account its requests for movement authorization. This registration is done in two steps. The first step is a pre-registration by the rail node that the train leaves before it even crosses it (a token is placed in the *STC2RBC* place and arrives in *WaitingTrains* after crossing the *PreRegistration* transition – see Figure 13). As soon as the train arrives on the first block of the track, it sends its position to the RBC to confirm its registration (A token is placed in the *EVC2RBC* place and the *TrainPosition* place after firing of the *PositionReport* transition - see Figure 13).

For the sake of simplicity, each sending of a position is interpreted as a request for authorization of movement. Also when crossing the *PositionReport* transition, a token is placed in the *MAReq* place, which is then processed by the RBC based on the list of trains modeled by the *Managed Trains List* place (see Figure 13). It is important to note the difference in priority between transitions *Update Position* (priority P_HIGH) and *CreatMA* (priority P_HIGH+1). This difference ensures that the position update is performed before the movement authorization is managed. Indeed, in the case of simultaneous validation, the transition with the highest case of same priority, the firing is random based on token semantic.

When the train arrives on the final block of the line, it makes a route request to the controller of the reached node (STC for a station or JTC for a junction). This sub-function is modelled by the green part of Figure 13. When the train is connected with another controller, and has left the line, it sends a disconnect request to the RBC that is processed by crossing the *Disconnect* transition, which removes the train's record in the *Managed Trains List* place (part in red in Figure 13).

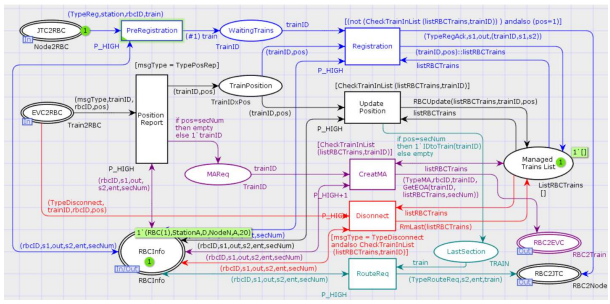

Figure 13: RBC Model

5.3. Modeling of the JTC component

In this section, we are interested in the modeling of the controller of a junction node. This controller is responsible for the dynamic routing of trains crossing the node. Given the train's destination station, the JTC searches in its base for a route with all resources (switches and track sections) available. For this purpose, the potential crossing routes are predefined. When the controller receives a route request from the train, it assigns an available route with respect to the arrival line at the node and the destination station. Once the route has been allocated, the route is established by specific switch controllers. These controllers are not modelled in this study and are considered part of the environment. At the end the JTC notifies the train of the route availability (place *JTC2EVC*) and notifies the RBC (place *JTC2RBC*) of the destination line in order to trigger the pre-registration of the train. All of these operations are modelled in Figure 14.

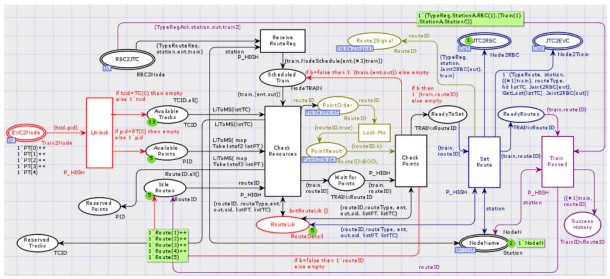

Figure 14: Model of the junction component

ISBN 978-88-85741-31-7; Bruzzone, Dauphin-Tanguy and Junco Eds

6. CHECKING MODEL PROPERTIES

In order to add generic components in the library, it is first necessary to check them. The necessary verifications are primarily the good engineering properties of PN models: liveness, boundedness, no deadlock. But any formal verification under CPN tools requires the prior construction of the reachability graph (also called occurrence graph) of the system in question. The difficulty in verifying these properties is related to the modular approach used for the design. Each component being a module with interface places, it can only evolve thanks to the interaction with its environment. There are several modular verification techniques available. The simplest technique consists in placing the expected answers in the interface places of the server components. This method is simple but does not offer the necessary quality guarantees. Indeed, it can also require to initialize internal places of the model with values according to what we want to verify. Since there are several interface places, it is necessary to anticipate all the tokens necessary for the evolution of the model. Unfortunately, in this case, this can lead to the simultaneous validation of several transitions. As the token game semantics of PN interpretation leads to a random firing of the model transitions, this can lead to test scenarios that do not conform to the design logic of the component in relation to its use in a railway application.

Another method is the calculation of colored invariants. But this method is complicated in the general case of CP-nets and even more High-Level Petri Nets. CP-nets classes such as Well-formed PN have characteristics that allow colored invariants to be calculated. But it is more difficult to model complex systems with this type of PN, which imposes restrictions on the colors and functions used in system modeling (Xie et al., 2017).

A third method is to use the compositional verification technique. For component modeling, this technique consists of constructing a reduced model of the component environment. Although interesting, the challenge is to build scale models that are compatible with the test scenarios. If some models are too small the verification may not be complete.

To address these difficulties, we have developed a new form of modular verification: reactive modular verification. This form of verification is based on work done on reactive nets (Eshuis and Dehnert, 2003). Our objective is to reduce the size of the reachability graph of a component and its environment. For that, one distinguishes the semantics of interpretation of the component to be checked from that of its environment. The principle is to keep the models of the other components as they are but to interpret them with reactive semantics. The component to be checked keeps the token game semantics. This allows the environment to react by giving quickly acknowledges to the requests made to it by the component. Figure 15 illustrates the application of state space construction with the use of both semantics. In this illustration, we have given process A (that represents the component to be checked), a token game semantics and process B (that represents the environment of component A) a reactive semantics. We can see the resulting reachability graph corresponding to part in red on the Figure 15. With the reactive semantics applied to process B, the interleaving of the firings of the transitions of the two processes is eliminated. The black part of the reachability graph is therefore deleted with the reactive semantics applied to process B. This allows reducing the state space without having to abstract the environment model in order to reduce the global model (of the component to be checked and its environment) and then its state space. As a result, considering the state space corresponding to the red part of Figure 15, one can conclude that process A is live, bounded and reversible.

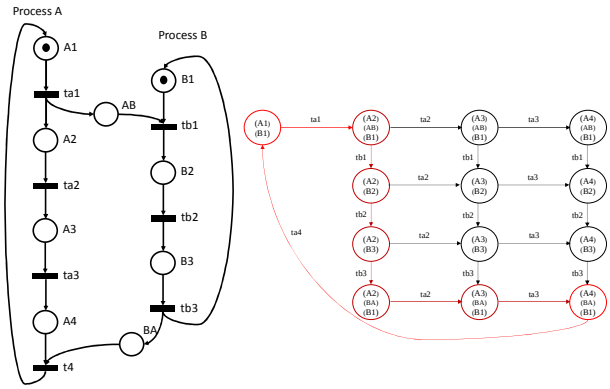

Figure 15: Principle of reactive modular verification

To implement this differentiated semantics in CPN Tools, it is sufficient to use different priorities between the component to be checked (default normal priority - P_NORMAL) and its environment. For example, to check the train model, we gave maximum priority (P_HIGH) to the RBC and JTC components (compare priority of transitions of EVC model in Figure 12 and RBC model in Figure 13). The priority P_INSTATION is less than the priority P_HIGH but greater that the priority P_LOW.

7. ILLUSTRATIVE CASE STUDY

In this section, we will illustrate how to use the previous generic components to model a railway system. To do this, we will use as a case study the example presented by Figure 16.

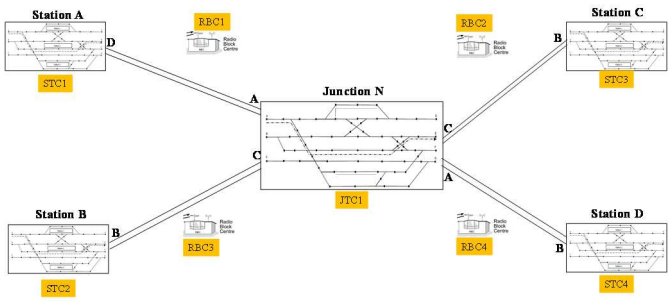

Figure 16: Example of railway network

ISBN 978-88-85741-31-7; Bruzzone, Dauphin-Tanguy and Junco Eds

It has 4 stations named station A to station D and a junction for the interconnection of the lines leading to these stations. Let us suppose that two trains are initially located in station A and they must go to station C at different times. To reach station C, they must go through junction N. The exit door from station A to go to N is door D. Station A and junction N are connected by Line 1 managed by RBC1. To reach Station C, trains must exit through door C and take Line 3 controlled by RBC2.

7.1. The global model

Following the decomposition of Figure 3, one abstracts the railway control system as composed of two EVC to model the trains (EVC1 and EVC2), one RBC for line 1 (RBC1) and another one fore line 3 (RBC2) and JTC1 to model the controller of junction N.

In the global layer (Figure 17), each instance is modeled as a substitution transition and is parametrized by its configuration place. As an example, the configuration place of EVC1 instance is called *T1info* and the configuration place of EVC2 is called *T2info.* In order to have a more compact representation, a unique configuration place with a compound colored token in a form of (4) to represent both the train identifier and the necessary train attributes is used.

$$(\text{Identifier}, \text{Attribute } 1, .., \text{Attribute n}) \quad (4)$$

As an example, the initial marking of *T1info* is:

$$M_0(T1info) = 1`(Train(1), StationA, StationC) \quad (5)$$

Train(1) is the identifier of the first train. *StationA* and *StationC* define respectively the departure and the destination station of *Train(1)*. In, the same way is defined the parameters of each RBC.

$$M_0(RBC1Info) = 1`(RBC(1), StationA, D, NodeN, A, 20) \quad (6)$$

Equation (6) means that *RBC(1)* controls the line from *StationA gate D* to junction *N gate A*. This line is composed of 20 blocks. This last parameter is essential for the RBC to know when a train has reach the end of the line that it controls in order to request to the JTC a route for the train (corresponds to the firing of transition *RouteReq* in RBC model in Figure 13) .

The interface places enable to model the communication from one type of components to another type of components. For example, in Figure 17 the place *EVC2RBC* models the communications from the EVC instances to the RBC instance. In this case study, it is not possible for EVC to request itself a route to the junction N. The structure of colored tokens put in the interface places enables to define the concrete participants of the communication. As an example, it is worth to notice the initialization of place *JTC2RBC* in Figure 17.

$$M_0(JTC2RBC) = 1`(TypeReg, StationA, RBC(1), (Train(1), StationA, StationC)) \quad (7)$$

This initialization enables us to simulate the request of *StationA* that requests *RBC(1)* to perform a preregistration of *Train(1)* that is arriving on the line controlled by *RBC(1)*. It also gives to *RBC(1)* all the data of the itinerary of *Train(1).* One can also notice that the initial marking of *T1BM* is a list of data that are like *(Line(1), (Inline,1,TC(1))* or *(Line(0), (InStation, 20, TC(1))*. *Line(1)* means that it defines a train position on Line(1). As an example *(Line(1), (Inline,1, TC(1))* means that the train is on the block 1 of the Line(1) and will reach NodeN by its track section TC(1). *(Line(0), (InStation, 20, TC(1))* means that *Train(1)* is in a node (station or junction) on the track section *TC(1)* and is arrived by the block 20 of the line before the node. So the list allows modeling the sequence of the train positions for simulation or the construction of its reachability graph. The global layer model can be easily modified by connecting/disconnecting components to the corresponding interfaces.

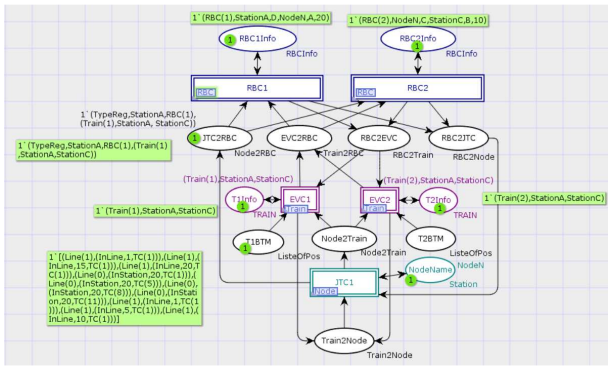

Figure 17: An example of global model based on instantiation of generic components.

7.2. Reachability graph construction for formal verification of a generic component

In order to perform properties' verification, it is necessary to build the reachability graph of the component to be checked with its environment. Considering the global PN model of section 7.1, let us suppose that one wants to check EVC component (more precisely EVC1). To apply the reactive modular verification, let us modified the priority of all the transitions of RBC1, RBC2 and Node N, from P_NORMAL to P_HIGH (a way to perform reactive semantics). The priority of transitions of the EVC components remain at P_NORMAL (a way to keep the token game semantics) except the transitions *InBlock* and *InNode* that keep the priority P_LOW.

To give an idea of the complexity of this verification, we have constructed the reachability graph of the global model of our case study by just considering the evolution of one train (*Train(1)* corresponding to EVC1) going from station A to station C. This corresponds to the initial marking in Figure 17. One can note that the *T2BTM* place that models *Train(2)* positions is not marked indicating that it remains immobilized.

Table 1 shows the results of this evaluation in terms of size and duration. They show the interest of building

ISBN 978-88-85741-31-7; Bruzzone, Dauphin-Tanguy and Junco Eds

reachability graphs using a mix of the two semantics (reactive semantics for the environment). These results have been obtained on an Intel CORE i7-5600U CPU at 2,60 GHz, with 16 GO of RAM.

Table 1: Comparison of the size of the reachability graphs constructed with the two semantics

	Reachability graph			
Type of semantics	**Number of nodes**	**Number of arcs**	**Number of dead markings**	**Duration (in seconds)**
Token semantics	5414	15042	35	13
Mix of the two semantics	38	37	1	1

As indicated by the fact that the number of arcs is equal to the number of states minus one, in the case of reactive semantics, the reachability graph is linear. The last state is blocking, modeling the arrival at station C. With the same initial marking and the use of token semantics for all the global model, the reachability graph is much larger. One can especially note that it has three times more arcs than nodes indicating the strong interleaving of the firing of transitions due to this semantics. This example illustrates the great efficiency of the construction of reachability graphs using reactive semantics for the environment.

The obtained reachability graph shows that our model is correct since the train has effectively followed its itinerary. In the obtained scenario, all the transitions of the EVC have been fired at least once (once for the transitions corresponding to the travel inside the node N, and several times for the others), proving that the global models is L1-live (Murata, 1989). In fact the modification of the global model with a transition that resumes the initial marking of this case study from de dead-marking obtained when *Train(1)* reaches station C, one shows that the EVC model is live and bounded.

Applying this approach to each generic component, proves that all our generic components are live and bounded.

8. CONCLUSIONS AND PERSPECTIVES

This work made it possible to propose a decentralized control architecture for the complete automation of rail system control. The objective is to go far beyond the automation of train operation as envisaged by the autonomous train concept. It showed that it was possible to develop a system modeling approach based on generic components placed in a library. The developed components correspond to our points of view that will be refined through standardization. They do not integrate all the complexity of modelling such systems. For example, we did not take into account the modeling of a train's operating modes, which include dozens of modes. We have not modelled train speed regulation that would require the use of a formalism such as hybrid PN (David and Alla, 2005). In this context, the question will arise of having a formalism that allows the characteristics of the two types of PN to be integrated.

The use of reactive semantics is an important way to reduce the combinatorial explosion of reachability graphs. The proposed approach allows proving the correctness of each generic component building a kind of global model with a distinction of the component to be checked and its environment. However, in order to check some properties such as the absence of possibility of collisions between two trains, it is necessary to develop another approach of verification that takes into account a more global model with several instances of trains.

REFERENCES

Christos G. Cassandras, & Lafortune, S., 2008. *Introduction to discrete event systems*. NewYork (USA), Springer.

Christensen, S., & Mortensen, K. H. Design/CPN ASK-CTL Manual, version 0.9, 1996. *URL http://cpntools.org/_media/documentation/askctl manual. pdf.[checked on 2013-12-16]*.

David, R., & Alla, H, 2005. *Discrete, continuous, and hybrid Petri nets* (Vol. 1). Berlin: Springer.

Eshuis R, Dehnert J. 2003. Reactive Petri Nets for Workflow Modeling. *Appl Theory Petri Nets 2003*, vol. 2679, pp. 295–314.

European Railway Agency. 2016. ERTMS/ETCS System Requirements Specification (SUBSET-026) v3.6.0

Jensen, K., & Rozenberg, G, 2012. *High-level Petri nets: theory and application*. Springer Science & Business Media.

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.

Lusby, R. M., Larsen, J., Ehrgott, M., & Ryan, D. M. (2013). A set packing inspired method for real-time junction train routing. *Computers & Operations Research*, 40(3), pp. 713-724.

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

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.

Xie, Y., Khlif-bouassida, M. & Toguyeni, A., 2017. Well-formed Petri Net Based Patterns for Modeling Logic Controllers for Autonomous Trains. Proc. of IMAACA2017, Barcelona (Spain), pp. 25–34

Yin, J., Tang, T., Yang, L., Xun, J., Huang, Y., & Gao, Z., 2017. Research and development of automatic train operation for railway transportation systems: A survey. *Transportation Research Part C: Emerging Technologies*, 85, 548-572.