# Taking advantage of some complementary modelling methods to meet critical system requirement specifications

F. Defossez[1], P. Bon[1] & S. Collart-Dutilleul[2]
[1]*INRETS, ESTAS, Villeneuve d'Asq, France*
[2]*LAGIS, École Centrale de Lille, France*

## Abstract

This paper aims at showing how it is possible to combine the advantages of high-level Petri nets and the *B* method in order to design safety applications. In the railway critical software domain, safety requirements are obviously severe. Indeed, the passing from an informal specification to a formal one is a crucial point in critical software development. High-level Petri nets combine three important features: a graphical representation, a dynamic behaviour and an abstraction of the treatments. The *B* method allows one to pass from an abstract specification to a concrete implementation. We propose an approach that integrates the structuring and modelling of the system behaviour by means of coloured Petri nets from semi-formal specifications and the generation of a *B* abstract specification from this Petri net.
*Keywords: railway critical systems, safety, formal methods, model translation, high-level Petri nets, B method.*

## 1 Introduction

The technological progress of safety automation in railway systems involves a growing complexity of functional safety requirements. Thereby, this leads to the use of some technical tools for analysis and command synthesis that are more and more complex and efficient in order to respect the requirements. Moreover, the introduction of new European standards for railway safety has led one to reconsider critical system requirement specifications modelling.

In such a context, formal methods seem to be one of the most adapted solutions. Indeed, their use in process development increases the understanding of the requirements and clarifies their expression. They also allow the verification and validation of the specification and make implementation easier. Nevertheless, their notation may be complex and difficult to understand.

In this paper, we put forward the hypothesis that the UML notation, Petri nets and the *B* method are widely used to model railway systems. Among other applications, they are used to model and validate real-time distributed railway systems. We focus on these methods, first in underlining their strengths and drawbacks, then in showing how they can be complementary. This is the purpose of the second part of this paper.

The aim of the article is not to design a new specification notation, but to compose existing ones to benefit from their complementary strengths. Indeed, these skills can be used together in order to increase the reliability of a railway application. Thus, formal methods can compensate for a lack of mathematical foundation of some models. The third part of this article describes a method found in the literature aimed at generating a formal *B* specification from UML diagrams.

Finally, we present our method which proposes an automatic translation of a high-level Petri net into *B* abstract machines. First, it consists of designing the Petri net from the extraction of key words from the specifications. Then, we upgrade the obtained Petri net by means of refinements. Eventually, the abstract machines are built in order ending with a classical *B* process.

## 2 Tools used to model railway systems

In this section, we focus on the three methods, in underlining their strengths and drawbacks and in showing how they can be complementary.

### 2.1 UML

The UML notation has become an industrial standard for the object-oriented modelling of software applications [1]. It is used to identify the requirements and describe the system. Its graphical nature makes discussions easier for the different actors of a project.

The UML notation makes it possible to model an application according to an object view, by means of several different diagram types. Thus, each diagram allows a particular view of the system. As a result, the diagrams allow a deep analysis and understanding of the system architecture and implementation details, as well as system functioning and operational features.

### 2.2 Petri nets

The Petri net formalism is a well-known graphical executable technique for the specification and analysis of concurrent discrete-event dynamic systems [2]. It is a relevant tool for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic or stochastic.

As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behaviour of systems. Having this rich mathematical background with precise semantics, it is proposed as a communication method enabling specialists of the domain, who are not accustomed to formal methods, to share their knowledge.

There are several extensions of this model: timed Petri nets, stochastic Petri nets or high-level Petri nets are some examples. This last extension is used in our method described in Section 4. Indeed, a realistic modelling of a complex system often requires one to reason about the nature of the tokens and about their transformation. With this aim in view, high level Petri nets allow the annotation of the net by means of a first order language. So tokens are expressions of the language and the transformations from one state to another are described by the formulas annotating the transitions. To sum up, high level Petri nets handle structured tokens and are annotated with a first order language. We use particularly coloured Petri nets [3], based on a functional language.

### 2.3 The *B* method

The *B* method, introduced by J. R. Abrial [4], is a formal method for the development of specifications and their refinements to an implementation. It is a collection of mathematically based techniques for the specification, design and implementation of software components. As a result, it is able to manage strong operating constraints applied to rail systems, such as CENELEC standards. Moreover, the *B* method seems to be the most appreciated method in the industrial world for railway critical software development such as METEOR [5].

The abstract machine is the basic element of *B* development. It models a system described by a set of data or variables and by the associated operations that modify their state or their value. An abstract machine is composed of:

- data declarations:
    - parameters,
    - variables,
    - constants,
- an invariant, that consists of a predicate on the previously declared elements and gives the types,
- a definition of the initial state,
- operations that define the actions modelling the state changes.

The *B* tools allow us to generate automatically the proof obligations for each abstract machine. The refinement process is a set of successive transformations from the initial model that aims at clarifying the abstract representation into a concrete model. At the last refinement, called the implementation, we obtain a secure software.

## 3 From UML to *B*

As explained previously, a benefit of using UML is its status as an international standard and its widespread use in the software industry. Nevertheless, it is only a semi-formal modelling tool. On the other hand, the *B* method is a formal method, already used in the railway industry, but its notation may be complex and difficult to understand. Moreover, a model analysis requires mathematical skills which are unusual in industry.

This section reminds us of a method presented in the literature [6] aimed at combining the advantages of UML and of the *B* method in order to design safety applications. During the critical software development process, safety requirements must be traced from informal specification to code generation.

This method aims at transforming a semi-formal modelling (UML) to a formal specification (*B* method) which enables them to be traced. In fact, it is able to develop safety critical systems by providing a semantic for a subset of UML and a systematic translation to the *B* method.

## 4 From specifications to formal implementation

We present our method to consider the translation from a high-level Petri net into the *B* method in order to integrate the methodology in a global issue of safety requirements. The process consists of covering a part of the software development. We propose a method of graphical description in order to obtain a formal specification as abstract machines from informal specifications. In order to obtain a more accessible specification, the method is built on four steps:

- modelling: permits one to obtain a graphical model of the specifications,
- interpretation: annotates the graph with natural language in order to complete the model,
- formalisation: transforms the annotations to obtain a high-level Petri net,
- translation: translates the Petri net into abstract *B* machines.

We apply this specification method to a case study presented in the next section.

### 4.1 Railway case study

In order to present the main aspects of the two formalisms and to show how they can be complementary, we chose a case study presented in [7]. Let us introduce a railway network specified by the following general rules:

1. the network is closed loop composed of seven elementary tracks,
2. two trains run in the same traffic direction,
3. two trains can't be on the same track,
4. there must be a free track between two trains.

In [7], this problem is specified with an elementary Petri net. This Petri net is quite heavy: it is composed of 21 places, 14 transitions and 84 arcs. This example underlines a limit in the use of elementary Petri nets to model complex problems. Thereby, in such a case, it is necessary to use high level Petri nets.

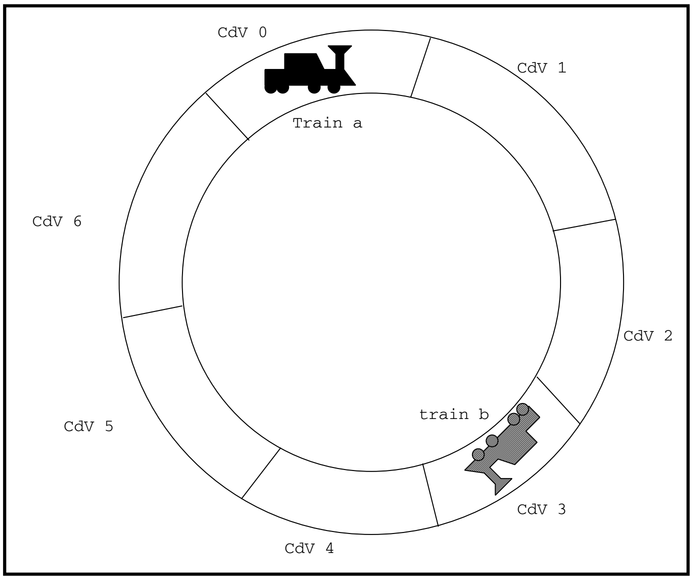

Figure 1: Schematic representation of the case study.

## 4.2 Modelling by means of a Petri net

In the first step of our method, the system described in the specifications is represented graphically as a Petri net. The main point of this step is the respect of the functional safety requirements. The identification of elementary actions, data of the system, constraints to respect and dynamic aspects permits one to structure the model.

Indeed, we have to collect as much information as possible with the Petri net in order to simplify the annotations in natural language and, as a result, the mathematical formulas which will follow them. We try to identify four kinds of words in the specifications, that each correspond to a precise component of the Petri net:

- actions, modelled with transitions,
- data, characterised by the state of the places,
- constraints to respect, described by annotations,
- dynamic aspects (events management), modelled by transitions and arcs.

## 4.3 Interpretation of the specifications

The second step of our method consists of labelling the graph obtained in the modelling phase. These annotations, expressed in natural language, have to be as close as possible to the specifications. This step is linked to the modelling phase and they can be lead at the same time. Thus, we obtain directly the natural language annotated Petri net.

## 4.4 Modelling and interpretation of the specifications of the case study

As expressed in section 4.2, the transitions of the Petri net model the passing from a track $i$ to a track $i+1$. A place before this transition models the presence of a train in the track $i$, and another shows if the track $i+1$ is free and permits the passage from $i$ to $i+1$. Thereby we consider tracks as resources that the train uses when it is on a track.

Then we have to model the constraint aiming at keeping a free track between two busy ones. In order to respect this constraint, the place modelling the availability of the track $i$ is linked with the transition modelling the passage of a train from the track $(i-1) \mod 7$ to the track $i$. This transition is linked to the place modelling the availability of the track $(i-2) \mod 7$.

So we need 7 places modelling the presence or not of a train in a given track. There could be coloured tokens in these places permitting us to characterize the different trains. Each transition is annotated with a sentence that sums up the modelled action, and each place with a sentence that models the state of the system if it contains a token. The tokens, which model the presence of a train in a given track, are coloured. They specify the type of train ($ta$ or $tb$) that occupies the track. Finally, we obtain the annotated graph of Figure 2.

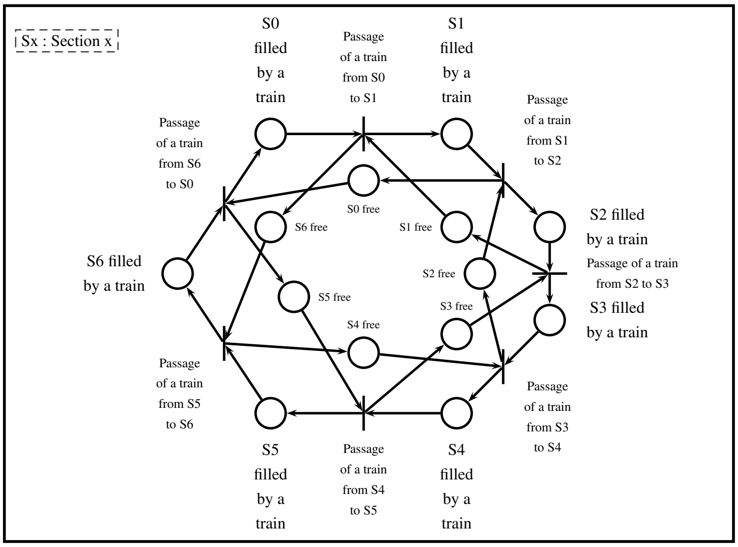
Sx : Section x
S0 filled by a train
Passage of a train from S0 to S1
S1 filled by a train
Passage of a train from S1 to S2
S2 filled by a train
Passage of a train from S2 to S3
S3 filled by a train
Passage of a train from S3 to S4
S4 filled by a train
Passage of a train from S4 to S5
S5 filled by a train
Passage of a train from S5 to S6
S6 filled by a train
Passage of a train from S6 to S0
S0 free
S1 free
S2 free
S3 free
S4 free
S5 free
S6 free

Figure 2: Annotations in natural language of the graphical model.

## 4.5 Formalisation

The third step of our method consists of converting the annotations of the Petri net into mathematical formulas. These formulas have to be as simple as possible and expressed as constraints. This step can be improved by means of a refinement of the Petri net: we can eventually gather some transitions (folding of the net), or integer new elements to improve the model.

The formalisation step permits us to obtain the coloured Petri net of Figure 3.

The initial marking of the Petri net indicates that the track 0 is occupied by a train $ta$ and the track 4 by a train $tb$. That implies that the markings of the tracks $1, 2$ and 5 are free. As explained previously, the Petri net can be reduced, if the tracks are not marked by simple tokens, but if we take their number into account. This allows a consistent simplification of the net: overall we obtain a Petri net only composed of 2 places and 1 transition that can be found in Figure 4. The marking becomes, for one place, numbers indicating the free tracks and, for the other, couples that indicate that one train is on a identified track.

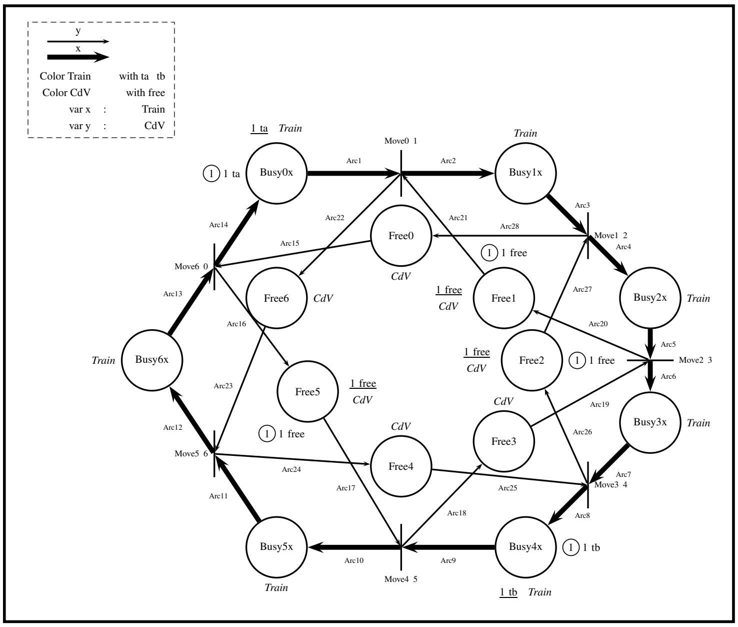

Figure 3: Coloured Petri net modelling of the case study.

## 4.6 Translation from the Petri net model to *B*

In this last step, an automatic transformation into a $B$ abstract machine is applied to the Petri net of Figure 4. Our approach consists of defining, first, a $B$ machine corresponding to the system modelled with a Petri net. Then, this machine is clarified by the definitions of the generic structural properties of Petri nets. Finally, the translation is completed with the integration of the dynamic properties of the Petri net.

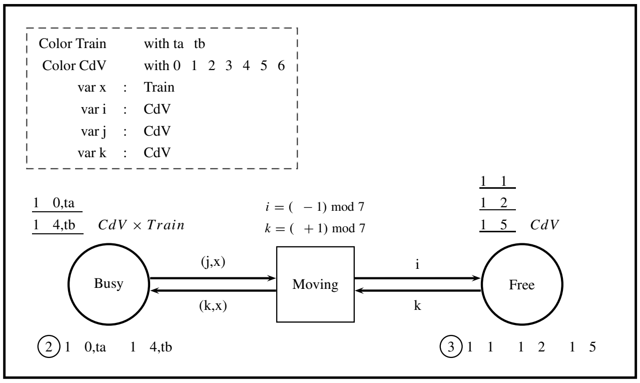

Figure 4: Simplified Petri net model of the case study.

```
SYSTEM RdPtrain
INCLUDES Multiset
SETS Trains = {ta, tb}
CONSTANTS CdV
PROPERTIES CdV = {elt | elt ∈ NAT ∧ elt ⩾ 0 ∧ elt ⩽ 6}
DEFINITIONS
       Var_Moving  ==  i, j, k, x ;
  Type_Var_Moving  ==  i ∈ CdV ∧ j ∈ CdV ∧ k ∈ CdV ∧ x ∈ Trains ;
  Guard_Moving  ==  i = ((j − 1) mod 7) ∧ (kk = (jj + 1) mod 7) ;
  ArcExp_Busy_Moving  ==  Ms_Empty(ColorF_Busy) ⩤ {(jj ↦ xx) ↦ 1} ;
   ArcExp_Free_Moving  ==  Ms_Empty(ColorF_Free) ⩤ {kk ↦ 1} ;
  ArcExp_Moving_Busy  ==  Ms_Empty(ColorF_Busy) ⩤ {(kk ↦ xx) ↦ 1} ;
   ArcExp_Moving_Free  ==  Ms_Empty(ColorF_Free) ⩤ {ii ↦ 1}
       Var_Moving  ==  i, j, k, x ;
  Type_Var_Moving  ==  i ∈ CdV ∧ j ∈ CdV ∧ k ∈ CdV ∧ x ∈ Trains ;
  Enabled_Moving  ==  ∃ Var_Moving.(Type_Var_Moving ∧ Guard_Moving
                    ∧  Ms_Subset(ArcExpr_Free_Moving, State_Free, Color_Free)
                    ∧  Ms_Subset(ArcExpr_Busy_Moving, State_Busy, Color_Busy))
VARIABLES
  State_Busy, State_Free
INVARIANT
  State_Busy ∈ MS(ColorF_Busy) ∧ State_Free ∈ MS(ColorF_Free)
INITIALISATION
     State_Busy  :=  Ms_Empty(ColorF_Busy) ⩤ {(0 ↦ ta) ↦ 1, (4 ↦ tb) ↦ 1}
  ||  State_Free  :=  Ms_Empty(ColorF_Free) ⩤ {1 ↦ 1, 2 ↦ 1, 5 ↦ 1}
EVENTS
  Op_Moving =
   SELECT Enabled_Moving
   THEN   ANY Var_Moving
    WHERE
         Ms_Subset(ArcExpr_Busy_Moving, State_Busy, ColorF_Busy)
     ∧   Ms_Subset(ArcExpr_Free_Moving, State_Free, ColorF_Free)
     ∧   Type_Var_Moving ∧ Guard_Moving
    THEN
         State_Busy :=  Ms_Add(Ms_Less(State_Busy, ArcExpr_Busy_Moving,
                        ColorF_Busy), ArcExpr_Moving_Busy, ColorF_Busy)
     ||   State_Free :=  Ms_Add(Ms_Less(State_Free, ArcExpr_Free_Moving,
                        ColorF_Free), ArcExpr_Moving_Free, ColorF_Free)
    END
   END
END
```

Figure 5: Abstract machine corresponding to Figure 4.

High-level Petri nets handle multi-sets. In order to translate a high-level Petri net into $B$, we have to specify an abstract machine which defines these multi-sets, their properties and their associated operations. Concisely, the machine that precisely gives the multi-sets is not presented in this paper. At the end of the process, described in [8], we obtain the abstract machine of Figure 5.

## 5 Conclusion

We described in a study case an approach aiming at using high-level Petri nets within the framework of the specification of dynamic systems. The systematic generation of $B$ specifications from the Petri net model allows us to consider the use of the tools linked to the $B$ language in order to continue the conception process. Our method combines strengths of two existing formal methods.

We want to integrate this method into a more global safety analysis process. Some works have been done to take temporal requirements into account [9]. These describe how a critical system, such as a level-crossing, can be modelled with time Petri nets in order to evaluate and validate its safety. We aim at building a global methodology from the modelling of temporal requirements of critical railway systems to their automatic implementation by means of formal methods.

## References

[1] Unified modelling language version1.4. Technical report, OMG, 2001.

[2] Murata, T., Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 1989.

[3] Jensen, K., *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (volume 1)*. Springer-Verlag, 1992.

[4] Abrial, J.R., *The B Book - Assigning Programs to Meanings*. Cambridge University Press, 1996.

[5] Behm, P., Benoit, P., Faivre, A. & Meynadier, J.M., METEOR : A successful application of B in a large project. *Proceedings of FM'99: World Congress on Formal Methods*, pp. 369–387.

[6] Boulanger, J.L., Bon, P. & Mariano, G., From UML to B - a level crossing case. *COMPRAIL*, pp. 351–359, 2006.

[7] Genrich, H., *Predicate / Transition nets*. Springer-Verlag, pp. 3–43, 1991.

[8] Bon, P., *Du cahier des charges aux spécifications formelles : une méthode basée sur les réseaux de Petri de haut niveau (In French)*. Thèse de doctorat, Université des Sciences et Techniques de Lille, 2000.

[9] Defossez, F., Collart-dutilleul, S. & Bon, P., "Formal methods and temporal safety requirements: a level crossing application". *FORMS/FORMAT*, Braunschweig, Deutschland, pp. 351–359, 2007.