Journal of Universal Computer Science, vol. 19, no. 1 (2013), 2-24
submitted: 18/10/10, accepted: 28/12/12, appeared: 1/1/13 © J.UCS

From a Solution Model to a B Model for Verification of Safety Properties

Philippe Bon
(Univ Lille Nord de France,"59000 Lille, France
IFSTTAR, ESTAS
philippe.bon@ifsttar.fr)

Simon Collart-Dutilleul
(Univ Lille Nord de France,F-59000 Lille, France
IFSTTAR, ESTAS
simon.collart-dutilleul@ifsttar.fr)

Abstract: In the context of safety requirement engineering, model transformation is a task of interest. Indeed, it allows us to keep all the requirements while switching from one point of view to another. The presented work assumes that a valid solution has been found and proposes an approach in order to build a valid implementation. As some fine dynamic properties are integrated into the specification, high-level Petri nets are used to specify and verify the solution. Then, considering an industrial railway context, the transformation of the Petri net model in order to provide an input to a B process is considered. This last consideration leads to a proposition of a systematic direct transformation of the Petri net model into abstract B machines. The approach is illustrated by a theoretical railway example. The limitations of this approach are discussed at the end of the paper and some prospects are detailed.
Key Words: Petri nets, B formal method, modelling languages translation, safety critical system, railway transport
Category: J.6, J.7

1 Introduction

The aim of this paper is to describe the work to be performed when a fine behavioural specification has to be assessed by an expert. The methodology is presented, integrating the usual constraints of the railway systems, which will serve as an instance of industrial context. Let us point out that in this industrial context, the code generated by "Atelier B" is considered safe when the B model is proved to be safe. For this reason, the safety proof is restricted to providing a B model able to be assessed.

Petri nets have both the power of mathematics and the explicit graphical representation of critical mechanisms, such as parallelisms, synchronizations, choices and mutual exclusions. The rigorous underlying mathematical model is useful for providing formal proofs of some needed properties. Methodologies based on the UML modelling usually fail to provide formal proofs, because UML is only a semi-formal language. Let us point out two other interesting characteristics of Petri nets.

The first one is that it gives, in contrast to state charts, an explicit representation for synchronization. This is really an advantage when the use of a resource in a mutual exclusion structure has safety consequences. The second one is that Petri nets are able to provide dynamic specifications such as direct computation of minimum cycle durations and functioning margins [Sifakis 1977]. These last results are more difficult to achieve using the B method [Abrial 1996] for example.

Nevertheless, most of the requirements are difficult to express and to assess using formal methods. In some industrial areas, such as railway systems, the use of human expertise cannot be avoided [Defossez et al 2010]. A graphical model is really an advantage for an industrial expert who may not be familiar with mathematical formalisms. Moreover, modelling power is needed in order to provide a complex, but concise, model which will allow the expression of the know-how of the expert. Considering this point of view, high-level Petri nets provide a very strong modelling power, whereas they contain all the mathematical properties of ordinary Petri nets [Jensen and Rozenberg 1991].

This leads to the following problem. When the model has been validated by an expert, using some formal proofs or not, how can it be translated into a model which can be verified with respect to the specifications and automatically implemented? More precisely, the translation has to be proved in such a way that the expert assessment is preserved. In the railway context the B method is a formal method which is accepted as a good formalism for assessment. The B method can be processed until an executable code is produced. Consequently, the problem to be solved is the translation of a high-level Petri net language into B formalism.

The first section presents the high-level Petri nets abbreviation. The second part of this work deals with the B method. The third section is devoted to the main contribution of this paper. It corresponds to a building methodology of translation from high-level Petri nets to B abstract machines. The last section proposes some further ideas for research before concluding.

2 Requirement engineering motivations

The aim of this section is to provide an overview of the global approach (figure 1). It started with the elicitation phase which corresponds to the phase where all requirements are identified in the informal specification.

Then, the requirements are modelled in order to be analysed. During the requirement analysis, Petri net may be used in order to build a formal behavioural analysis. The model of the requirements may be used to perform the synthesis of a valid control [Collart Dutilleul et al.06], [Declerck and Guezzi 2009].

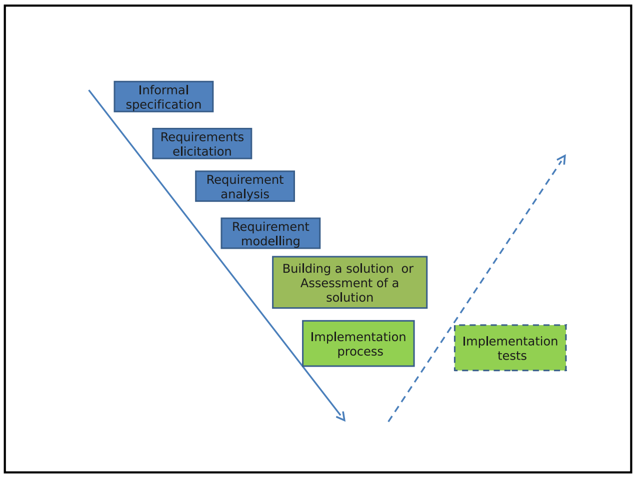

Figure 1: Diagram of the requirement engineering approach

Another way of using the requirement model is to check systematically the correctness of a would-be solution. In this case, a formal tool may be used to make a comparison between the behaviour of the would-be solution model and the requirement model [Defossez et al 2010].

However, at the end of the analysis phase, the design process is expected to deal with the specification of a valid solution.

Before the execution of the implementation task, a functional validity by an expert of the domain is interesting. In this case, a compact readable graphical model is useful. High-level Petri nets have both the quality of formal modelling and language power. When the solution validated by the expert is expressed with high-level Petri nets, as in [Philippi06], then the problem of translation of this model into an implementation one has to be faced.

This is precisely the subject of this paper: the problem is to provide a valid input into the process of implementation solution building. As the efficiency of the B process for producing an implantation is well-known, the translation of high-level Petri net into abstract B machines is really an interesting challenge.

3 Petri Net model

Petri nets were developed by C. A. Petri [Petri 1962] in order to model concepts of asynchronous and concurrent actions. Petri net theory allows modeller to specify dynamic behaviour of a system but also to understand and assess it. In this section, we start to present the basic theory and then we present high-level Petri nets, and more precisely, coloured Petri Net.

3.1 Place/transition Petri net

Petri nets are used in order to model the behaviour of discrete dynamic systems [Murata 1989]. A Petri net is a particular class of directed graphs with an initial state called *initial marking.* A Petri net is a bipartite graph. It has two types of nodes: places and transitions. Arcs link only a place to a transition or a transition to a place. Graphically, transitions are represented by bars or boxes and places by circles.

The use of elementary Petri nets to model complex problems is limited because of the size of the model. Thereby, in such a case, it is necessary to use high-level Petri nets. The size of the elementary Petri net is due to the fact that tokens cannot be differentiated.

3.2 High-level Petri net

As briefly mentioned below, in elementary Petri nets token cannot be differentiated. However, realistic modelling often needs to discuss the nature and transformations of tokens. To allow this, high-level Petri net can be labelled by a first-order language. The tokens become language expressions and transformations, from one state to another, are described by formulae labelling transitions. The marking is a multi-set of tokens and transitions firing corresponds to a multi-sets transformation. In short, high-level Petri nets handle structured tokens and are labelled by a first-order language. Several forms of high-level Petri net can be distinguished. The most common are:

- predicates/transitions Petri nets [Genrich 1987], based on first-order logic,
- coloured Petri nets [Jensen 1992], based on a functional language,
- algebraic Petri nets [Reisig 1991], based on an equational language.

To reduce the size, modeller can also use extensions of Petri net as Numerical Petri Nets developed by Symons [Symons 1978].

Only the coloured Petri nets are presented here. But before introducing them, some basic notion must be defined. First-order languages and associated definition are now presented.

3.2.1 First-order languages

A language definition consists in giving a mechanism of *sentence* building using an alphabet of *symbols* and assigning a *sense* to these sentences.The term *language syntax* is used for sentence building and *semantic* for sense assigning. In order to define first-order languages intuitively, a current mathematical language is used.

Then, let us consider a set of symbols of *variables*, a set of symbols of *functions* and a set of symbols of *predicates*. These sets are assumed to be disjoint from each others. To each symbol of functions (predicates), a positive integer called *arity* is associated. Symbol of *constants* are symbol of functions with an arity equal to 0. In a first-order language, *expressions* are built recursively, from constants and symbols of variables, with symbols of functions. For instance, in current mathematical language, $2 * x + f(y)$ is an expression built from constant 2 and variables x and y, with function symbol f (arity 1), $+$ and $*$ (arity 2, in unfixed notation with a priority of $*$ on $+$).

Likewise, an *atom* is built from expressions with symbols of predicates. For example, $2{*}x{+}y > z{-}3$ is an atom built from expressions $2{*}x{+}y$ and $z{-}3$, with predicate symbol $>$ (arity 2, in unfixed notation). Finally, *formula* is classically built from atoms with quantifiers $\exists$ and $\forall$ and logical connectors as conjunction $\wedge$, disjunction $\vee$, negation ,.... So, $(x = 3) \vee \exists y((x + y > 2) \wedge (x - y < 3))$ is a formula. An expression is said to be *closed* if there is no variable inside and a formula is closed if all of its variables are in a quantifier field. For instance, previous formula is not closed because x is free (i.e. no quantified by an $\exists$ or a $\forall$).

The *interpretation* of a first-order language consists in:

- associating, to each constant, a value in the *interpretation domain*,
- associating, to each symbol of function, a value function in the interpretation domain,
- associating, to each symbol of predicate, a *relation* (in other words, a boolean function).

Then, expressions and formulae are classically interpreted. For instance, on the interpretation domain of rationals, with the interpretation of $+$ and $*$ as addition and multiplication, the expression $2{*}4{+}5$ can be evaluated as 13. Likewise, with the interpretation of predicates $=, <$ and $>$ as equality and classical order relations, formula $\exists x \exists y((x + y > 2) \wedge (x - y < 3))$ is true. Let us note that interpretation depends on the interpretation domain. Formula $\exists x(2 * x - 1 = 0)$ is true in the rational domain, but false in the integer domain. After that informal presentation of first-order languages, some associated notions are more formally defined.

3.2.2 Notation and terminology

Now, syntax and the interpretation of a first-order language are formally defined.

Definition 1. Let V be a set of variables, Ω a set of symbols of *functions*, and Π a set of *predicates*. To each predicates and symbol of functions, a positive integer called *arity* is associated.

(i) The couple (Ω, Π) is called a *signature*.

(ii) An *expression* (or *term*) V built on Ω is:

- an arity 0 function (i.e. a *constant*),
- a variable v from V,
- a construction $f(e_1, ..., e_n)$ where f is a function symbol with arity n and $e_1, ..., e_n$ are expressions.

The set of expressions L is called an *algebra*. An expression without variables is said to be *closed*.

(iii) An *atom* built on (L, Π) is a construction $p(e_1, ..., e_n)$ where p is a predicate symbol with arity n and $e_1, ..., e_n$ are expressions. An atom without variables is said to be *closed*.

(iv) A *formula* is either:

- an atom,
- a construction $(F), F \wedge G, F \vee G, \neg F, F \Rightarrow G$ or $F \Leftrightarrow G$ where F and G are formulae,
- a construction $\exists x(F)$ or $\forall x(F)$ where x is a variable and F a formula (where there is no sub-formula as $\exists x(G)$ or $\forall x(G)$)

The set Ψ of formulae is called a *first-order language*.

(v) In a construction $\exists x(F)$ (resp. $\forall x(F)$), variable x appearing in F is said to be *linked* by the existential (resp. universal) quantifier. A variable without any links is said to be *free*. A formula without free variables is said to be *closed*, and a formula without quantifiers is said to be *free*.

(vi) A *theory* T is a set of closed formulae.

In the following, except contraindication, letters x, y, z, u, v, w denote variables, a, b, d values, f, g, h function symbols, p, q predicate symbols, e, c expressions, A, B, C atoms and F, G formulae.

The *substitution* notion is also often used in the theory of declarative language. This notion is also useful to interpret expressions with non-explicitly quantified variables. Intuitively, a substitution consists in replacing a variable with an expression. Formally, it is defined as follows:

Definition 2.

(i) A *substitution* $\sigma = [x_1/e_1, ..., x_n/e_n]$ is an application from the set of variables V to the set of expressions L as:

– $\sigma(x_i) = e_i$ for $i = 1, ..., n$,

– $\sigma(v) = v$ for all others variables.

(ii) The substitution notion is extended to an endomorphism on the set of expressions, more formally: $\sigma(f(e_1, ..., e_n)) = f(\sigma(e_1), ..., \sigma(e_n))$.

(iii) The composition $\sigma_1 \circ \sigma_2$ of two substitutions is defined by $(\sigma_1 \circ \sigma_2)(e) = \sigma_1(\sigma_2(e))$.

(iv) The application of the substitution to a free formula is defined as:

– $\sigma(p(e_1, ..., e_n)) = p(\sigma(e_1), ..., \sigma(e_n))$

– $\sigma(\neg F) = \neg\sigma(F)$

– $\sigma(F \ op \ G) = \sigma(F) \ op \ \sigma(G)$ where *op* is one of these boolean operators $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$

(v) A substitution $\eta = [x/a]$ where a is a constant is called *assignation* (or *valuation*).

(vi) A substitution $\alpha = [x/v]$ where v is a variable is called *renaming*.

(vii) The application of an assignation $\eta = [x/a]$ to a quantified formula replaces all *free* occurrences of x by a :

– $\eta(\exists x(F)) = \exists x(F)$

– $\eta(\exists v(F)) = \exists v(\eta(F))$ for each variable v different of x

– same for $\forall$.

After these definitions, coloured Petri nets can be defined.

3.2.3 Coloured Petri nets

A coloured Petri net [Jensen 1992] is a classical Petri net with a set of colours in order to distinguish tokens. The expression power of this type of Petri net allows us to model real systems. Coloured Petri nets are based on a functional language, where the typing notion is very important. Then, a type, limited to a finite set, is associated to each place. This type is called colour of the place. So, coloured Petri nets are based on a typed first-order language. There is a set of domains, and symbols of functions are interpreted as functions of values in these domains. It is important to notice that a coloured Petri net can always be unfolded into a place/transition Petri net. The formal definition of a coloured Petri net is given below:

Definition 3. A coloured Petri net is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ as

(i) Σ non-empty and finite set of types, called **colours**,

(ii) P is a finite set of places,

(iii) T is a finite set of transitions,

(iv) A is a finite set of arcs, as: $P \cap T = P \cap A = T \cap A = \emptyset$,

(v) N is the **node** function, defined from A to $P \times T \cup T \times P$,

(vi) C is the **color** function, defined from P to Σ,

(vii) G is the **guard** function, defined, from P to expressions, as:

$$\forall t \in T : [Type(G(t)) = \mathbb{B} \wedge Type(Var(G(t))) \subseteq \Sigma],$$

(viii) E is an **expression of arcs** function, defined from A to expressions, as:

$$\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$$

where $p(a)$ is a place of $N(a)$,

(ix) I is an **initialisation** function, defined from P to closed expressions, as:

$$\forall p \in P : [Type(I(p)) = C(p)_{MS}].$$

The reader can refer to [Jensen 1992] in order to have more details on the above definition. To define precisely coloured Petri net behaviour, some notions are mandatory. First, the definition of variables and expression is given:

Definition 4. – $\forall t \in T : Var(t) = \{v | v \in Var(G(t)) \vee \exists a \in A : v \in Var(E(a))\}$.

– $\forall (x_1, x_2) \in (P \times T \cup T \times P) : E(x_1, x_2) = \sum_{a \in A(x_1, x_2)} E(a).$

$Var(t)$ is called the set of variables of t when $E(x_1, x_2)$ is called the *expression* of (x_1, x_2).

Now, the definition of *transition link* is introduced:

Definition 5. A transition link t is a function b defined on $Var(t)$, as:

(i) $\forall v \in Var(t) : b(v) \in Type(v)$,

(ii) $G(t) < b >$, where $G(t) < b >$ is predicate denoting the evaluation of the guard of t by link b.

The set of all transition t links is denoted by $B(t)$.

Complementary notions as token, binding element, marking and step are now defined:

Definition 6. – A token is a couple (p, c), where $p \in P$ and $c \in C(p)$, TE denotes the set of all tokens.

- A binding element is a couple (b, t), where $t \in T$ and $b \in B(t)$, BE denotes the set of all binding elements.
- A marking is a multi-set based on TE. The initial marking M_0 is the marking obtained by the evaluation of the initialisation expressions:

  $$\forall (p, c) \in TE : M_0(p, c) = (I(p))(c).$$

  $\mathbb{M}$ denotes the set of all marking.
- A step is a non-empty finite based on BE. $\mathbb{Y}$ denotes the set of all steps.

Now, the *step validation*, which allows us to describe the behaviour of a coloured Petri net, can be defined:

Definition 7. A step Y is enabled by a marking M if and only if the following property is checked:

$$\forall p \in P : \sum_{(t,b)\in Y} E(p, t) < b > \leqslant M(p).$$

Let Y be an enabled step for marking M. Then:

- if $(t, b) \in Y$, transition t is enabled for marking M for link b, by extension, (t, b) is also said to be enabled for M,
- if $(t_1, b_1), (t_2, b_2) \in Y$ and $(t_1, b_1) \neq (t_2, b_2)$, (t_1, b_1) and (t_2, b_2) are concurrently enabled, and then t_1 and t_2 are also concurrently enabled,
- if $|Y(t)| \geqslant 2$, then t is itself concurrently enabled,
- if $Y(t, b) \geqslant 2$, then (t, b) is itself concurrently enabled.

This notion of step allows us to express the possible simultaneity of transition firing.

Definition 8. When a step is enabled, it can be fired and it then changes marking M_1 into M_2 as:

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p, t) < b >) + \sum_{(t,b)\in Y} E(t, p) < b >.$$

The first sum represents consumed tokens while the second one represents produced tokens. M_2 is said to be directly reachable from M_1 by the occurrence of Y and it is formally noted as follows: $M_1[Y > M_2$.

Finally, the notion of firing sequence occurrence, which allows us to define reachable marking, is defined as follows:

Definition 9. An occurrence of a finite sequence is a sequence of markings and steps noted:

$$M_1[Y_1 > M_2[Y_2 > M_3 \dots M_n[Y_n > M_{n+1}$$

as $n \in \mathbb{N}$ and $M_i[Y_i > M_{i+1}$ for all $i \in 1..n$. M_1 is said to be the start marking and M_{n+1} final marking. Positive integer n is said to be the step number of sequence occurrence, or also length. In notation, halfway markings can be omitted:

$$M_1[Y_1Y_2 \dots Y_n > M_{n+1}$$

Definition 10. A marking M' is reachable from M if and only if there is an occurrence of a finite sequence with M as start marking and M' as final marking, i.e. if and only if for $n \in \mathbb{N}$, there is a step sequence as:

$$M[Y_1Y_2 \dots Y_n > M'.$$

M' is said to be reachable from M in n steps. The set of all reachable markings from M is denoted $[M >$.

As mentioned in the beginning of this paragraph, a coloured Petri net can always be unfold into a place/transition Petri net. The equivalence rules are not detailed here, the reader can refer to [Jensen 1992] in order to have them.

3.3 Railway illustration for coloured Petri nets

In order to illustrate the different tools presented in this paper, a theoretical railway example is used as a case study. This example is given by [Genrich 1991]. The case study is described by the general following rules:

c1 :	Railway network is composed of consecutive elementary parts, called CdV_i with $i \in [0..6]$
c2 :	trains run in the same traffic direction.

Two safety rules are now introduced:

c'1 :	Two trains cannot be on the same track at the same moment,
c'2 :	there must be a free track segment between two trains.

Finally, a particular case of railway network is considered:

c"1 :	The railway network is a closed loop of seven tracks, numbered from 0 to 6.

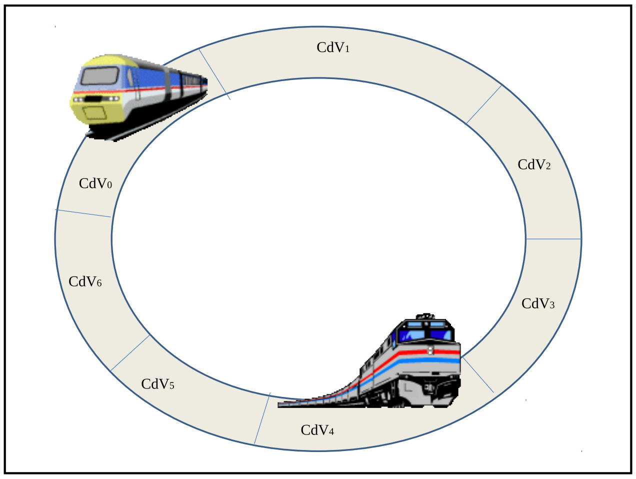

Figure 2: Schematic representation of the case study

Figure 2 gives a schematic representation of the case study described below. The 6 tracks are represented, on the figure, by CdV_i with $i \in [0..6]$.

In [Genrich 1991], the case study is specified with an elementary Petri net. This Petri net is quite heavy: it is composed of 21 places, 14 transitions and 84 arcs. The use of high-level Petri nets is necessary to reduce the size of the model. In order to do this, two types of token ta and tb, representing the two types of train, are introduced. The coloured Petri net of figure 3 models the theoretical railway example. 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 (multi-set $1'ta$ in place $Busy0x$ and $1'tb$ in $Busy4x$). That implies that the markings of the tracks $1, 2$ and 5 are free (multi-set $1'free$ in places $Free1$, $Free2$ and $Free5$).

The Petri net can be reduced if the tracks are not marked by simple tokens. If the track numbers are taken into account, a consistent simplification of the net is possible: 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 an identified track. The passage of a train from one track to the next is modelled by the transition and the guard gives the condition to respect safety requirements (c'1 and c'2). The last requirement (c"1) is modelled by markings, which give tracks numbers, and by the transition guard, $i = (j-1) \bmod 7$ and $k = (j+1) \bmod 7$ which model the circuit.

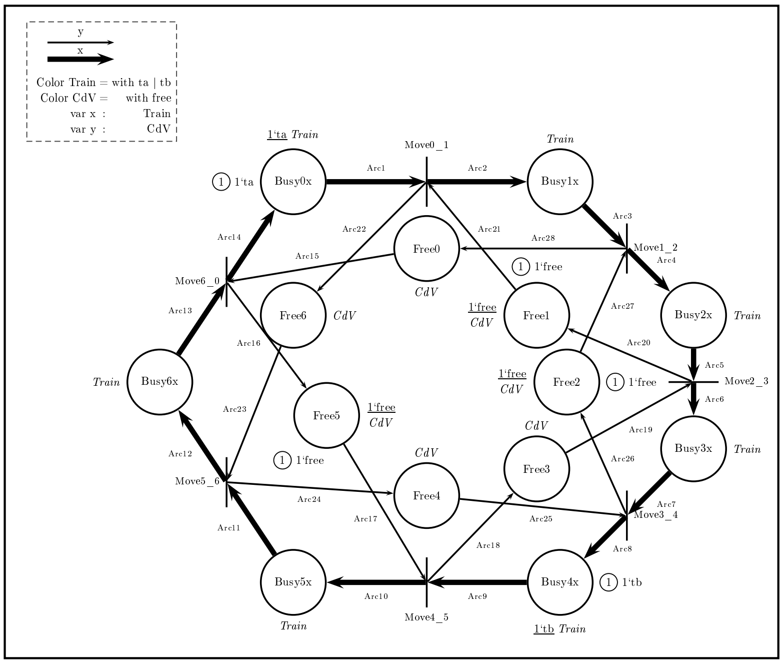

Figure 3: Coloured Petri net model of railway case study

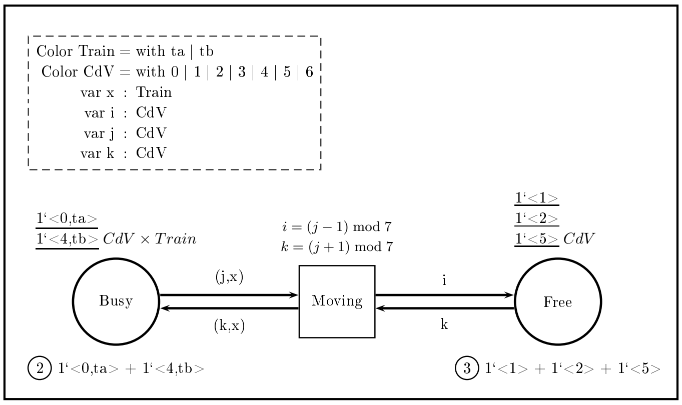

Figure 4: Simplified colored Petri net model of the case study

4 *B* method

The B method, which was developed by Jean-Raymond Abrial [Abrial 1996], is a formal model-oriented method such as Z and VDM. These methods are based on two complementary models: the static one describes the system entities and their associated states, and the dynamic model describes allowed changes of state by actions defined on entities. The dynamic model allows us to describe properties which have to be checked before and after action. These properties are expressed by classical logic predicates on entities and states. In a specification based on models, the system state is described by the set of couples (predicates,expressions) where the predicate set models the static aspect. The description of state changes models the dynamic aspect. Models are built with three characteristic elements:

- **pre-condition** is defined by the set of states from which the state change is allowed,
- **operation** is composed of the list of modifications to couples (predicate, expression),
- **post-condition** characterises valid states as ensuing from changes.

These notions are common for formal model-oriented methods. Nevertheless, in B notation, the notion of substitution replaces the notion of pre and post condition [Abrial 1996]. B also differs because it integrates the concept of refinement which makes incremental development from the specifications to the code possible in a single formalism. This formalism is called *the abstract machine notation*. Proof obligations are generated at each stage of the B development process in order to guarantee the validity of the refinement and the abstract machine.

As a result, it is able to manage strong design constraints applied to rail systems, such as CENELEC standards. Moreover, the B method seems to be an efficient method in the industrial world for railway critical software development, such as METEOR ([Behm et al 1999], [DaSilva et al 1992]).

4.1 Abstract machine notation

Modelling of data and their properties is based, in B language, on mathematical notation, essentially on the set theory. However, in the B set theory, the notion of typing is introduced. All the elements of a set are the same type. The principal data structures available are: sets, binary relations between sets, functions from one set to another and ordered lists of elements of a set. It can be noted that, in B, properties are expressed by formulae from calculus of first-order predicate with equality. That means the B language builds their predicates with classical propositional operators (and ($\wedge$), or ($\vee$) ...), but also with equality operator and quantified variables ($\exists x.P$ and $\forall x.P$).

The abstract machine (figure 5) is the basic element of a B development. It models a system described by a set of data or variables and by the operations associated that modify their state or their value. An abstract machine is composed of:

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

Then, an abstract machine models the behaviour of the specified system. Afterwards, this model is refined. An abstract machine is composed of different clauses representing the data of the specified system. In B, constants and sets represent unchanging data of the system. Each machine is defined by its name and can have parameters. The logic properties on these parameters are specified in the clause **CONSTRAINTS**. The sets (reps. the constants) are specified in the clause **SETS** (resp. **CONSTANTS**) and their logical properties in the clause **PROPERTIES**. The clause **VARIABLES** gives machine variables which represent variable elements of the system. As constants, variables are defined by a conjunction of predicates in clause **INVARIANT**. This clause gives the properties that the values of the variables have to satisfy at any time. Finally, variables are valued in the clause **INITIALISATION**.

In B language, there is an explicit clause, called **DEFINITION**, to specify some abbreviations.

Definition 11. A definition introduces an abbreviation, eventually with parameters, for a predicate, an expression or a substitution. A definition can be used in other clauses of the component. Each use of a definition is replaced by the corresponding text, where formal parameters take the place of real parameters. A definition can only be used in the component where it is defined.

Now, the static part of a system can be modelled. The dynamic one is specified by operations which correspond to actions to be performed by the system. In the operations, another fundamental notion of B language is used: the notion of *generalised substitution*.

4.2 Generalised substitutions

The generalised substitution notation allows us modelling services (actions) which had to be performed by the system. It is a key notion of the B approach.

Definition 12. A generalised substitution is a predicate transformer, which, when it is applied to predicates characterising data before a given service activation, produces the predicates characterising data after the service achievement.

In other words, if S is a substitution and P a predicate, then $S[P]$ is the predicate obtained applying S to P.

The generalised substitutions are an extension of the elementary substitution defined as follows:

Definition 13. if x is a variable and e is an expression, then the elementary substitution $x := e$, applied to a predicate P, transforms P in a predicate P', obtained by substituting e to all free occurrences of x in P.

The reader can find all the substitutions, used in the B language, and their description in [Abrial 1996] and [Lano 1996].

4.3 Refinement and implementation

As mentioned before, the B method uses the concept of refinement which provides incremental development from the specifications to the code in a single formalism. The refinement process is the set of successive transformations of the initial model. These transformations are made in order to put in concrete form the manipulated structures and solve indeterminism in order to obtain a software written in a common programming language. There can be several levels of refinement and the last one is called implementation.

Figure 5 gives a view of a refinement N from an abstract M and the differences between the two components.

Finally, the B method can be processed by using tools[1] which allow us to generate automatically the proof obligations for each abstract machine. At the last refinement, called the implementation, we obtain a safe software.

---

[1] two software platforms are available and give a set of automatic tools in order to develop real systems:

- the Atelier B, http://www.atelierb.eu/
- the B-Toolkit, http://www.b-core.com/btoolkit.html

```
MACHINE M(p)
CONSTRAINTS
C
SETS
St
CONSTANTS
k
PROPERTIES
B
VARIABLES
v
DEFINITIONS
D
INVARIANT
I
INITIALISATION
T
OPERATIONS
y <- op(x)      =
             PRE   P
             THEN
               S
             END
...
END
```

```
REFINEMENT N
REFINES
M
SETS
St1
CONSTANTS
k1
PROPERTIES
B1
VARIABLES
w
DEFINITION
D1
INVARIANT
J
INITIALISATION
T1
OPERATIONS
y <- op(x)      =
             PRE   P1
             THEN
               S1
             END
...
END
```

Figure 5: "Generic" abstract machine and its refinement

5 Transformation by construction

Now, a transformation algorithm is applied to the Petri net model in order to translate it into a B abstract machine. The proposed approach begins by building the abstract machine corresponding to the considered Petri net model. The second step adds some complementary information ensuing from the structure. Finally, the behavioural properties of the Petri net model are introduced in the B machine [Bon 2000].

5.1 Multi-set specification

As high-level Petri nets marking corresponds to a multi-set, a preliminary task is dedicated to the specification of multi-sets. This part does not define all the properties associated with multi-set. The reader can refer to [Bon 2000] to have more information.

In B, for typing reasons, it is not possible to define an abstract machine which takes into account all types of multi-sets. So, multi-sets and their associated properties and operations are specified with parametrised definitions. These definitions have to be included as necessary. Figure 6 gives the abstract machine, only composed of B definitions, specifying multi-sets.

$$
\begin{array}{l}
\textbf{MACHINE } \mathit{Multiset} \\
\textbf{DEFINITION} \\
\begin{array}{rl}
Ms(ss) & == ss \leftrightarrow NAT; \\
Ms\_Empty(ss) & == \{elt | elt : ss \times \{0\}\} \\
Ms\_In(elt, ms, ss) & == \exists n.(n \in NAT_1 \wedge (elt \mapsto n) \in ms) \\
Ms\_Subset(ms1, ms2, ss) & == \forall elt.(Ms\_In(elt, ms1, ss) \\
& \quad \Rightarrow ms1(elt) \geqslant ms2(elt)) \\
Ms\_Add(ms1, ms2, ss) & == \lambda ee.(ee \in ss | ms1(ee) + ms2(ee) \\
Ms\_Less(ms1, ms2, ss) & == \lambda ee.(ee \in ss | ms1(ee) - ms2(ee)
\end{array} \\
\textbf{END}
\end{array}
$$

Figure 6: Abstract specification of multi-sets and their associated properties and operations

Then the structure and properties to be fulfilled by the Petri nets component are presented.

5.2 Systematic transformation from a Petri net into an abstract *B* machine

First, the translation of the structural aspects of the Petri net model is described. A state variable correspond to each place of the Petri net.

Then, the translation of a Petri net transition into a B operation is implemented. This transformation takes into account the fact that the impact of an operation on the state variable is the same as the corresponding transition firing.

5.2.1 Petri net structure transformation

A machine describes the structure of the Petri net modelling the considered system. It seems to be natural to translate the marking of each place by a state variable of the B abstract machine. As the marking is a multi-set of tokens and a token is a t-uple composed of s elements which are the colours of this place. For each place p a state variable $State\_p$ is produced. This variable is defined as a multi-set based on the color $ColorF\_p$ associated to the place p. Consequently for each place p the following scheme is introduced:

$$
\begin{array}{l}
\textbf{VARIABLES} \\
State\_p \\
\textbf{INVARIANT} \\
State\_p \in MS(ColorF\_p)
\end{array}
$$

As there are two different places on the considered railway example, it corresponds to:

VARIABLES
$State\_Busy$,
$State\_Free$
INVARIANT
$State\_Busy \in MS(ColorF\_Busy)$
$\wedge\ State\_Free \in MS(ColorF\_Free)$

It can be noticed that, as the first part of translation shows, the transformation consists of a simple syntaxical transformation from Petri net labels into B language. From the structural definition of a coloured Petri net, a transformation rule is applied. The previous part correspond to:

(i) Σ is a non-empty finite set of types, called **colors**

(ii) P is a finite set of places

(iii) T is a finite set of transitions

(iv) A is a finite set of arcs, as: $P \cap T = P \cap A = T \cap A = \emptyset$,

(v) N the **node** function defined from A to $P \times T \cup T \times P$

(vi) C is the **color** function defined from P to Σ

(vii) G is the **guard** function defined from P to expressions as:

$$\forall t \in T : [Type(G(t)) = \mathbb{B} \wedge Type(Var(G(t))) \subseteq \Sigma],$$

(viii) E is an **expression of arcs** function defined from A to expressions as:

$$\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$$

(ix) I an **initialisation** function defined from P to closed expressions as:

$$\forall p \in P : [Type(I(p)) = C(p)_{MS}].$$

To each color from Σ, a type of the abstract machine is associated. In the case study, the set of trains is enumerated and tracks are defined with an abstraction from naturals:

MACHINE $RdPtrain$
INCLUDES $Multiset$
SETS
$Trains = \{ta, tb\}$
CONSTANTS
CdV
PROPERTIES $CdV = \{elt | elt \in NAT \wedge elt \geqslant 0 \wedge elt \leqslant 6\}$
...

Sets P,T and A and function N do not explicitly appear in the abstract machine. In fact, the aim of this transformation is to capture the behaviour of the model and not the structure. In order to simplify the translation, the structure is not translated.

Functions C, G and E are specified by means of B definitions and, finally, the function I is specified in the **INITIALISATION** clause.

5.2.2 Behaviour transformation

The aim of this translation is essentially to capture the behaviour of the system. Before specifying the behaviour, some information on transition variables has to be introduced. The set of variables associated to a transition is composed by variables appearing in the transition guard and variables used by expression of arcs linked to the transition:

$$\forall t \in T : Var(t) = \{v | v \in Var(G(t)) \vee \exists a \in A : v \in Var(E(a))\}.$$

The list of variables is translated into a list of identifiers. These variables are typed by means of a definition with a typifying predicate. To each transition, a predicate $Enabled\_t$ denoting the transition t enabling in the current state is specified. In the case study, there is only one transition with two input arcs:

...
DEFINITIONS
...

$$\begin{array}{lll} Enabled\_Moving == & \exists Var\_Moving.(& \\ & & Type\_Var\_Moving \wedge Guard\_Moving \\ & \wedge & Ms\_Subset(ArcExpr\_Free\_Moving, State\_Free, \\ & & Color\_Free) \\ & \wedge & Ms\_Subset(ArcExpr\_Busy\_Moving, State\_Busy, \\ & & Color\_Busy)) \end{array}$$

...

When transition enabling conditions are specified, the operation allowing the Petri net evolution can be specified. As mentioned in paragraph 3.2.3, the token consumption (resp. production) is equivalent to a subtraction (resp. addition) of multi-sets. So, to each transition, an operation describing the marking evolution during the transition firing is specified. Finally, figure 7 gives the translation of the coloured Petri net model of the case study.

In this section the fundamental theory on which the translation methodology is based is presented. This translation is illustrated on an elementary railway example, but the proofs of the transformation are not presented. However, the mathematical aspects of the proof are developed in [Bon 2000].

In this contribution, the requirement engineering point of view is rather developed. Considering this last point of view, the most interesting contribution is the possibility of transforming an invariant I to be verified by a Petri net into a B abstract machine invariant. The use of automatic proof tools associated with

MACHINE *RdPtrain*
INCLUDES *Multiset*
SETS $Trains = \{ta, tb\}$
CONSTANTS CdV
PROPERTIES $CdV \quad = \quad \{elt \mid elt \in NAT \wedge elt \geqslant 0 \wedge elt \leqslant 6\}$
DEFINITIONS

$Var\_Moving \quad == \quad i, j, k, x\,;$

$Type\_Var\_Moving \quad == \quad i \in CdV \;\wedge\; j \in CdV \;\wedge\; k \in CdV \;\wedge\; x \in Trains\,;$

$Guard\_Moving \quad == \quad i = ((j-1) \bmod 7) \wedge (kk = (jj+1) \bmod 7)\,;$

$ArcExp\_Busy\_Moving \quad == \quad Ms\_Empty(ColorF\_Busy) \;\mathbin{<\!\!+}\; \{(jj \mapsto xx) \mapsto 1\}\,;$

$ArcExp\_Free\_Moving \quad == \quad Ms\_Empty(ColorF\_Free) \;\mathbin{<\!\!+}\; \{kk \mapsto 1\}\,;$

$ArcExp\_Moving\_Busy \quad == \quad Ms\_Empty(ColorF\_Busy) \;\mathbin{<\!\!+}\; \{(kk \mapsto xx) \mapsto 1\}\,;$

$ArcExp\_Moving\_Free \quad == \quad Ms\_Empty(ColorF\_Free) \;\mathbin{<\!\!+}\; \{ii \mapsto 1\}$

$Var\_Moving \quad == \quad i, j, k, x\,;$

$Type\_Var\_Moving \quad == \quad i \in CdV \;\wedge\; j \in CdV \;\wedge\; k \in CdV \;\wedge\; x \in Trains\,;$

$Enabled\_Moving \quad == \quad \exists\, Var\_Moving.(Type\_Var\_Moving \wedge Guard\_Moving$
$\quad \wedge \quad Ms\_Subset(ArcExpr\_Free\_Moving, State\_Free, Color\_Free)$
$\quad \wedge \quad Ms\_Subset(ArcExpr\_Busy\_Moving, State\_Busy, Color\_Busy))$

VARIABLES
$State\_Busy, State\_Free$
INVARIANT
$State\_Busy \in MS(ColorF\_Busy) \wedge State\_Free \in MS(ColorF\_Free)$
INITIALISATION

$State\_Busy \quad := \quad Ms\_Empty(ColorF\_Busy) \;\mathbin{<\!\!+}\; \{(0 \mapsto ta) \mapsto 1, (4 \mapsto tb) \mapsto 1\}$

$\| \quad State\_Free \quad := \quad Ms\_Empty(ColorF\_Free) \;\mathbin{<\!\!+}\; \{1 \mapsto 1, 2 \mapsto 1, 5 \mapsto 1\}$

OPERATIONS
$Op\_Moving =$
SELECT $Enabled\_Moving$
THEN **ANY** Var_Moving
WHERE

$Ms\_Subset(ArcExpr\_Busy\_Moving, State\_Busy, ColorF\_Busy)$

$\wedge \quad Ms\_Subset(ArcExpr\_Free\_Moving, State\_Free, ColorF\_Free)$

$\wedge \quad Type\_Var\_Moving \wedge Guard\_Moving$

THEN

$State\_Busy := \quad Ms\_Add(Ms\_Less(State\_Busy, ArcExpr\_Busy\_Moving,$
$\qquad ColorF\_Busy), ArcExpr\_Moving\_Busy, ColorF\_Busy)$

$\| \quad State\_Free := \quad Ms\_Add(Ms\_Less(State\_Free, ArcExpr\_Free\_Moving,$
$\qquad ColorF\_Free), ArcExpr\_Moving\_Free, ColorF\_Free)$

END
END
END

Figure 7: Abstract machine corresponding to the figure 4

the B framework may be of efficient assistance in this this difficult task. Focusing on the example described in this paper, there is a safety requirement forbidding that a train is on a track directly adjacent to an occupied track. This last property is difficult to verify directly on the Petri net model. The B expression of this requirement is as follows:

$$\begin{aligned}\forall(i,j).\ (&i \in CdV \wedge j \in CdV \wedge \\ &Ms\_In((i \mapsto ta), State\_Busy, Color\_Busy) \wedge \\ &Ms\_In((j \mapsto tb), State\_Busy, Color\_Busy) \\ &\Rightarrow (j-i) \bmod 7 > 1 \wedge (j-i) \bmod 7 < 6)\end{aligned}$$

This invariant takes into account that the railway example is circular and composed of 7 tracks. The proof of the above invariant, which can be provided by the B tools, allow us to claim the correctness of the Petri net model with regards to the considered safety requirement. B tools have an associated proving tool based on Hoare logic. This tool will generate proof obligations ensuing from the invariant. In some case, the tool has to be assisted by human expert in order to provide an addicted mathematical knowledge. Anyway, B tool experts use to formulate invariants in order to help the proving tool to succed.

6 Conclusion

Motivations of a mixed approach based on the use of different modelling tools in the context of safety requirements engineering is presented in the first part of this paper. This discussion put emphasis on the critical task which tackles the steps to perform from the requirements analysis towards a valid implementation on a real system. Actually there is a switching point where implementation considerations are introduced. Precisely, at this stage of the design process, the paper focuses on model transformation. This task may assist the designer on the way from analysis to implementation. Considering safety requirements in guided transports, transformation from high-level Petri nets into B abstract machines is considered. This is a way of keeping the same requirements, while switching points of views. Abstract B machines are a valid input of the analysis phase into the B process implementation synthesis.

Based on a simple example of section mutual exclusion railway problem, the high-level Petri net model powerfulness is illustrated. Moreover, the scalability and conciseness of the produced model are explained. Fundamental definitions of both Petri nets and B abstract machines are presented in such a way that a systematic translation can be introduced. This translation is presented and illustrated on the same example.

Now, starting from the example, integrating timetable constraints into the approach will provide a problem. From the point of view of the global engineering process (figure 1), defining an implementation parameter at the model analysis step is not correct. It just happens that high-level Petri nets use to handle finite domains. This does not correspond to the infinite number of values which can be assigned to a continuous variable belonging to an interval.

In the case of time constraints, the scientific literature provides some propositions introducing some interesting prospects [Bender et al 2008].

Moreover, the literature contains behavioural analysis results towards general continuous parameters [Dhouibi et al 2008]. The bridge to be built from the analysis process towards a valid implementation process does not only concern the time parameter. Obviously, informal specifications contain requirements concerning time, positions, dimensions, costs, speed, etc. They all correspond to values to be exactly defined at the implementation phase. From a theoretical point of view, dealing with these kind of constraints, the path from specification to implementation seems to exist. However, further research has to be conducted considering more practical aspects.

References

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

[Behm et al 1999] Behm P., Benoit P., Faivre A., Meynadier J. M.: "METEOR: A successful application of *B* in a large project", in Proceedings of FM'99: World Congress on Formal Methods, pp 369–387, 1999.

[Bender et al 2008] Bender D., Combemale B, Crégut X., Farines J. M., Vernadat F.: "Ladder Metamodeling & PLC Program Validation through Time Petri Nets", in Fourth European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA), Lecture Notes in Computer Science (LNCS), volume 5095, pages 121–136, Springer-Verlag, 2008.

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

[Collart Dutilleul et al.06] Collart Dutilleul S., Defossez F., Bon P.: "Safety requirements and p-time Petri nets: a level crossing case study", in IMACS multiconference on computational engineering in systems applications (CESA), pages 1118–1123, Bejin, Chine, 2006.

[DaSilva et al 1992] DaSilva C., Dehbonei B., Mejia F.: "Formal specification in the development of industrial applications: The subway speed control system", in FORTE'92 ,Proceedings of the IFIP TC6/WG6.1 Fifth International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, pages 199–213, 1992.

[Declerck and Guezzi 2009] DECLERCK P., GUEZZI A.: "Trajectory Tracking Control of a Timed Event Graph with Specifications Defined by a P-time Event Graph", in Positive Systems: Theory and Applications (POSTA 09), Lecture Notes in Computer Science (LNCS), volume 389, Springer-Verlag, 2009.

[Defossez et al 2010] Defossez, F., Collart Dutilleul, S., Bon, P.: "A formal model of requirements", in Open Transportation Journal, 2010, to be published.

[Dhouibi et al 2008] Dhouibi H., Collart Dutilleul S., Nabli L., Craye E.: "Using Interval Constrained Petri Nets for Reactive Control Design: A tobacco manufacturing application", in the International Journal for Manufacturing Science & Production, volume 9, number 3-4, pages 217–229, 2008.

[Genrich 1987] Genrich, H. J.: "Predicate/Transition Nets", in Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, Lecture Notes on Computer Science, volume 254, pages 208–247, Springer-Verlag, 1987.

[Genrich 1991] Genrich, H. J.: chapter "Predicate / Transition Nets", in High-level Petri Nets, volume 1, pages 3–43, Springer-Verlag, 1991.

[Ghezzi et al 1994] Ghezzi C., Morasca S., Pezzè M.:"Validating timing requirements for time basic net specifications", Journal of Systems and Software, volume 27, number 2, pages 97–117, 1994.

[Jensen 1992] Jensen, K.: "Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Vol. 1", Springer-Verlag, 1992, 234p.

[Jensen and Rozenberg 1991] Jensen, K., Rozenberg, R.:"High-level Petri nets", Springer-Verlag, 1991.

[Lano 1996] Lano, K: "The B Language and Method : A guide to Practical Formal Development", Springer Verlag London Ltd., 1996.

[Murata 1989] Murata T.: "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE vol. 77, N°4, pages 541–574, 1989.

[Petri 1962] Petri, C. A.: "Kommunikation mit Automaten", PhD Thesis, 1962, Kommunikation mit Automaten (in German).

[Philippi06] Philippi S.: "Automatic code generation from highlevel Petri nets for model driven systems engineering", in Journal of Systems and Software, volume 79, number 10, pp. 1444–1455, 2006.

[Reisig 1991] Reisig, W.: chapter "Petri Nets and Algebraic Specification", in High-level Petri Nets, volume 1, pages 137-170, Springer-Verlag, 1991.

[Sifakis 1977] Sifakis, J.: "Petri nets for performance evaluation", Measuring, Modeling, and Evaluating Computer Systems (Proceedings of the 3rd Symposium, IFIP Working Group 7.3), pages 75–93, 1977.

[Symons 1978] Symons, F.J.W.: "Modelling and analysis of communication protocols using Numerical Petri Nets", PhD Thesis, Dep. Elec. Eng. Sci, Univ Essex, Telecommun. Syst. Group Rep. 152, May 1978.