From UML to B – a level crossing case study

J.-L. Boulanger[1], P. Bon[2] & G. Mariano[2]
[1] *Universite de Technologie de Compiegne, HEUDIASYC, Compiegne cedex, France*
[2] *INRETS, ESTAS, Villeneuve d'ascq, France*

Abstract

The goal of this paper is to show how it is possible to combine the advantages of Unified Modelling Language (UML) and of the B method in order to design safety applications. We investigate how the Unified Modeling Langage (UML), can be used to formally specify and verify critical railways systems. A benefit of using UML is it status as an international standard (OMG) and its widespread use in the software industry. B is a formal method for the incremental development of specifications and their refinements down to an implementation. In the railway critical software domain, safety requirements are obviously severe. It is very important to keep requirements traceability during software development process even if the different used models are informal, semi formal or formal.
Keywords: B method, formal development, level crossing, software verification.

1 Introduction

In spite of progress carried out in software development, designing a complex system while respecting its safety requirements, remains very hard. During the critical software development process, safety and security requirements must be traced from informal specification to code generation. So we need to trace them in the different models: informal, semi formal or formal ones. We present a new method here to transform a semi formal modelling to a formal specification which enables them to be traced. This method will be applied to a railways case study, where safety requirements are very strict. We study a level crossing case study taking into account French particularities. This article is made up 3 parts. Firstly, we describe the case study. In the following part we present the principles of UML and we give a part of semi formal modelling of the level crossing. In the last part


WIT Transactions on The Built Environment, Vol 88, © 2006 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/CR060351

we give a short presentation of the B method, and then we describe a part of the formal model generation from the semi formal one.

2 The case study : level-crossing

To illustrate our approach, we will choose to design a level crossing. This example is inspired by [8]. The term level crossing, in general a crossing at the same level, i.e. without bridge or tunnel, is especially used in the case where a road crosses a railway; it also applies when a light rail line with separate right-of-way crosses a road; the term "metro" usually means by definition that there are no level crossings. Firstly, a single-track line, which crosses a road in the same level, is modelled. The crossing zone is named danger zone. The most important security rule is to avoid collision by prohibiting road and railway traffic simultaneously on level crossing]. The railway crossing is equipped with barriers and road traffic lights to forbid the car passage. Two sensors appear on the railroad to detect the beginning (train entrance) and the end (train exit) of the level crossing protection procedure. The level crossing is not in an urban zone this implies a sound signalisation. Traffic lights consist of a single flashing red light. When they are switched off, road users (drivers, pedestrians,) can cross. In the other case, the level crossing is closed and railway traffic has priority. At any time, guards can control the level crossing. In this case, the guards must do all they can to ensure the level crossing safety.

3 Semi-formal specification

3.1 UML

Born from the different object methods, like OMT or Booch & Jacobson, and normalised by the Object Management Group, UML has now become a standard to model systems. The UML notation makes it possible to model an application according to an object view. 9 different diagram types make this modelisation. Each diagram allows a particular view of the system. The reader interested by more details in syntaxic and semantic aspects can refer to the reference guide of UML [1]. Even if UML notation is a language in which models can be represented, it does not define the making process of these models. Nevertheless, several dedicated tools have strengthened the popularity of UML. These tools allow graphic notation and partial generation of the associated code and documentations. The UML notation is known by most computer scientists and is now used in several domains. Using UML class diagrams to define information structures has now become standard practice in industry. Recently, the critical application domains have used the notation and several questions exist around this use. By providing a rigorous semantic for a subset of UML and a systematic translation to the B method, we aim to propose a method, called **UML2B**, for developing safety critical system.

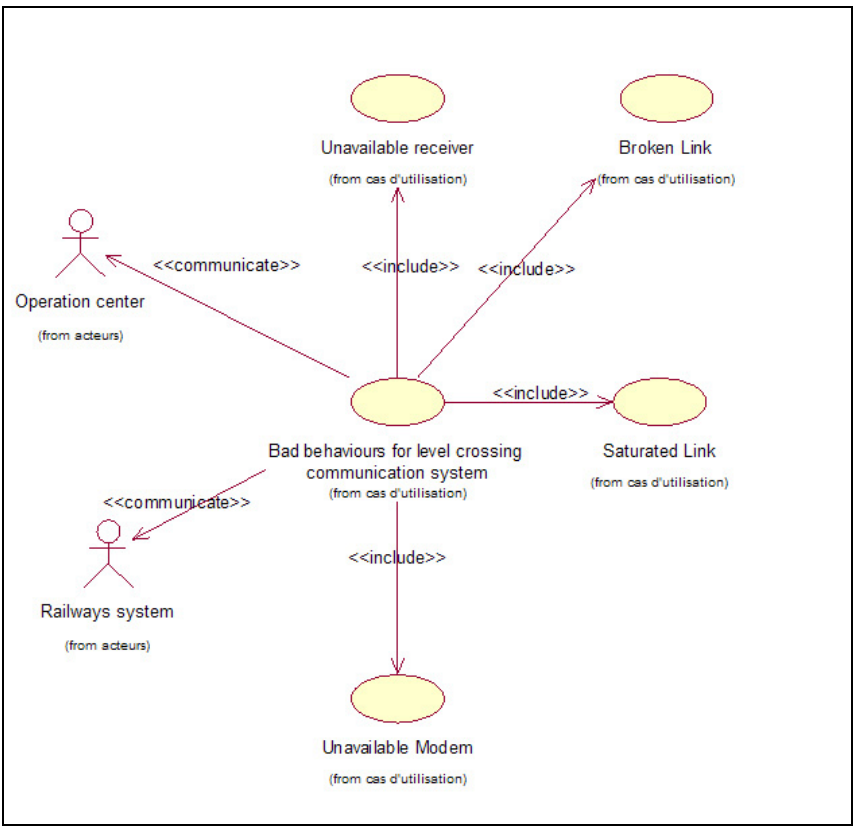

Figure 1: An usecase diagram.

3.2 Application

3.2.1 Environment and its interactions on the system

We describes the interaction between the physical environment and the level crossing system by some usecase diagrams (see an example in figure 1). An usecase diagram describes and traces the functional requirements of the system and describe how the system can and will be used. The usecase diagram gives an overview of the model. In UML, objects communicate with and send messages to each other. We uses sequence diagram (see figure 2) to describe how objects interact and communicate with each other. The focus is time.

3.2.2 Architecture of level crossing system

The level crossing system architecture is describe in the class diagram (see figure 3), which describes the relationships between classes and shows the logical view of a system (static view).

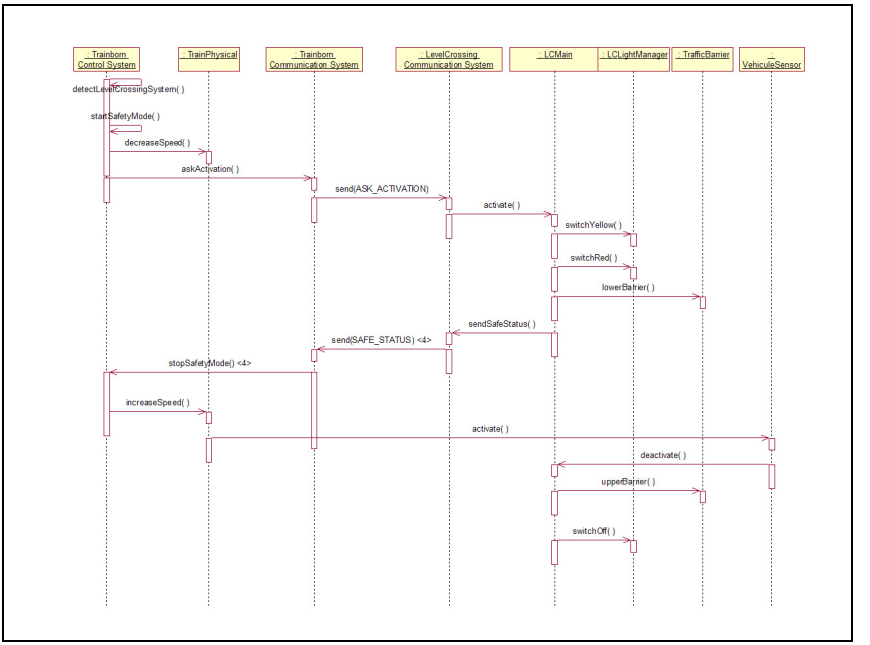

Figure 2: A sequence diagram.

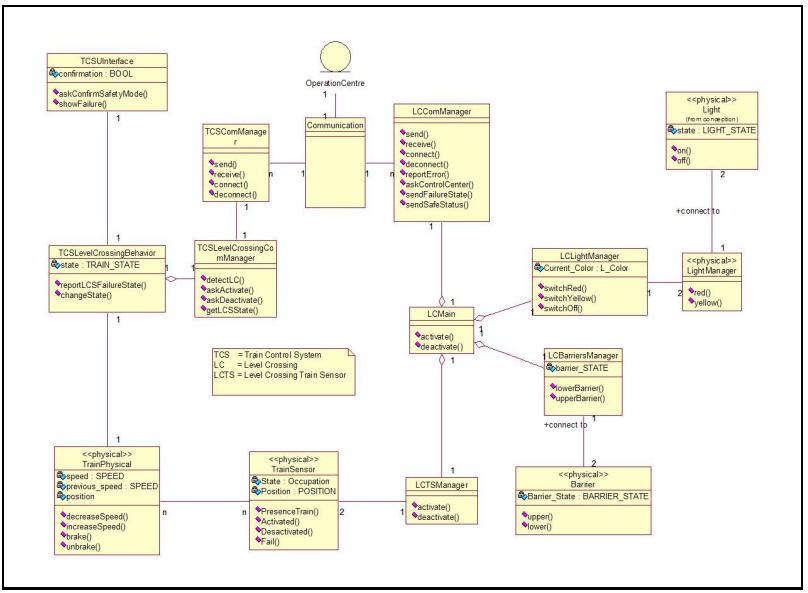

Figure 3: Class diagram.

3.2.3 Behaviour

In the next part of the paper, among the different possible diagrams, we'll use the state diagrams particularly adapted to reactive system modelling. A state is a condition in an object life while it satisfies some conditions, runs some actions or waits for some events. There are two specials states: initial state and end state. The Initial State is the state of an object before any transition. End States marks the destruction of the object whose state we are modelling. An event is a particular occurrence that can trigger a transition from a state to another one. A state diagram can representing the system behaviour. This type of diagram represents finite state automaton, under a graphical representation, linked by oriented arcs describing transitions. Statechart diagrams [6, 7], also known as State diagrams, are used to document the various modes ("state") that a class can go through, and the events that cause a state transition. The state-transitions graph formalism is not a UML innovation. It has often been employed in other contexts and a large consensus, from David Harel's works, exists around this notation. It introduces the description of possible sequences of states or actions which can occur to an element during its life. Such sequences arise from element reaction to discrete events.

3.2.3.1 Level crossing control system. Gates, lights and bell are managed by the level crossing control system (Figure 4). It will be activated as soon as a train approach the level crossing. The activation is produced by the railway traffic. The point, from which activation is produced, is called start of danger zone. A time allowed is needed between the activation and the arrival of the fastest train into the level crossing. This time, named train on line period, depends on maximal train speed into the level crossing. The moment the control system is activated, a sequence of orderly action is launched by the control system in order to empty the level crossing in time and protect it from road traffic. First, the lights switch on in order to stop the road traffic. At the same time, the bells ring (in a non urban zone). After a notice time, the barriers pull down. If there is no problem during the pulling down (i.e. pulling down made in the maximal temporal limits), the system is in safe mode and the train will have the right to cross. After the train crossing, the barrier pull up and the lights and bells switch off. The control system can at any time go to manual mode. In this mode a guard manages the level crossing security. This mode can be activating when a material problem occurs (no pulling down, no light switch on,).

3.2.3.2 Embedded system. The embedded system runs several actions when the train comes near a crossing level (Figure 5). When the train pass the start of danger zone, the embedded system asks to the control system an acknowledgement (ack), the embedded system gets into stand by and begin to brake in order to pull down the barrier in time. After this notice time, the control system sends its state to embedded one. If the level crossing is in safe mode, the embedded system stops the braking and restarts with its initial speed. An end-crossing sensor detects the train exit and starts the barrier pull up and the lights switch off.

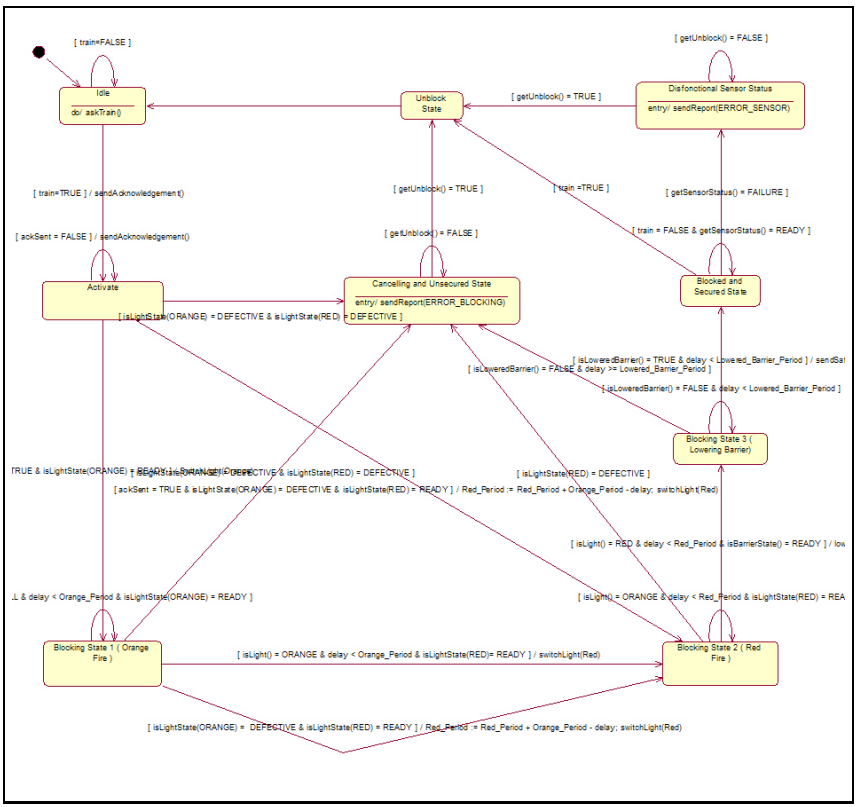

Figure 4: State transition diagram of level crossing control system.

4 Formal specification

4.1 The B method

The B method seems to be the most appreciated formal method in industrial world for railway critical software development such METEOR (see [3] or [5]). In fact, B method allows building a gate between mathematical modelling and informatics realisation. The B method due to J.R Abrial [2] is a formal method for the incremental development of specifications and their refinements down to an implementation. It is a model-based approach similar to **Z** and **VDM**. The B method covers all the software development process through a series of proved refinement steps. The software design in B starts from mathematical specifications(set description, first order logic and substitution). Little by little, through many refinement steps [9], the designer tries to obtain a complete and executable specification. This process must be monotonic, that is any refinement has to be proved coherent according to the previous steps of refinement. The B

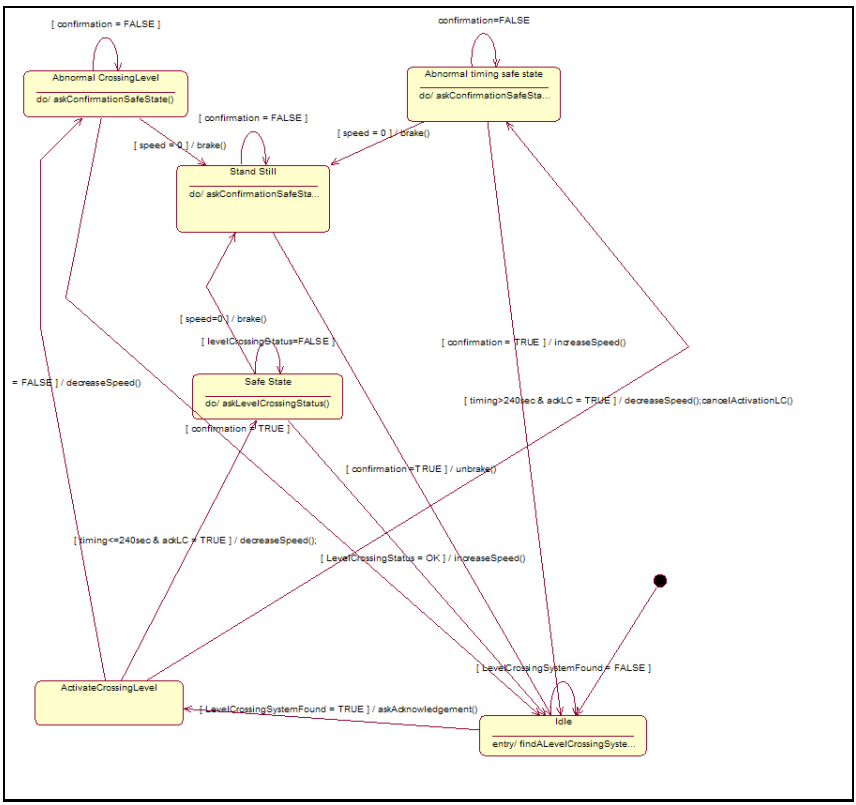

Figure 5: Embedded system running.

method allows generating a secured and proved code from formal mathematical expressions. The **abstract machine** is the basic element of a B development. It encapsulates some state data and offers some operations. The B tool allows us to generate automatically the proof obligations for each abstract machine. Generally speaking, the proof obligations will be more and more complex as concrete details are introduced. Then these proof obligation are discarded either automatically for the simple ones or in cooperation with the designer for the complex ones. So, at the last refinement called the implementation, we obtain a secure software which does not need to be tested. At this low level stage, it may be easily translated automatically to a programming language.

4.2 B specifications generated from UML diagrams

This section presents the B specification generated from UML diagram for the Level Crossing problem. For space constraints, we do not describe all the specifications. We fully presented the B specification generation processus in [4].

```
MACHINE TCS_LevelCrossingBehavior_0
SETS STATES ={Idle,SafeState,ActivateCrossingLevel,StandStill,
              AbnormalTimingSafeState, AbnormalCrossingLevel}
CONCRETE_VARIABLES current_state
INVARIANT  current_state : STATES
INITIALISATION current_state := Idle
OPERATIONS
change_state =
BEGIN
current_state :(
 current_state :STATES
&((current_state$0= Idle)=>current_state:{ActivateCrossingLevel,Idle})
...
&((current_state$0= StandStill)
   => current_state/:{SafeState, AbnormalTimingSafeState})
)
END
END
```

The previous B abstract component, called TCS_LevelCrossingBehavior_0, we introduced all states in set called STATES and an operation called "change_state". This abstract component is not deterministic since we use the operator $list\_var : (predicate)$ in the OPERATION clauses. This operator indicates that the list of variable become such that the predicate is true. In the next abstract machine, we implemented the abstract component "TCS_LevelCrossingBehavior_n". This abstract machine introduces the dynamic of the state diagram.

```
IMPLEMENTATION        TCS_LevelCrossingBehavior_n
REFINES               TCS_LevelCrossingBehavior_0
INVARIANT (current_state = StandStill) => (brake = TRUE)
  &       (current_state /= StandStill) => (brake = FALSE)
INITIALISATION        current_state := Idle
OPERATIONS
change_state =
CASE current_state OF
                EITHER  Idle THEN
                VAR bb IN
                        bb <-- detectLevelCrossingSystem;
                        IF      ( bb = TRUE )
                        THEN
                        BEGIN   current_state := ActivateCrossingLevel
                        ;       AskAcknowledgement
                        END
                        ELSIF   ( bb = FALSE)
                        THEN current_state := Idle
                        END
                END
....
END
END
```

5 Conclusions

With these two main parts, Semi formal modelling with UML and formal specification with B, this works show the first step that allow developing a completely computerised level crossing. This application will be totally proved and will guarantee an optimal safety. Of course the real development must be done, but

with some add-on, the method presented here allow considering such development. Such an example of future work, safety invariants can be derived from hazard analysis and coded in UML by using OCL constraints attached to classes or sets of associations to specify safety and operational invariants of reactive systems in a concise manner. We will translate these UML constraints (OCL) into invariants and control rules in the B model. The main difficulty to specify railway case study is the less of harmonisation between the different European systems. The level crossing modelling presented here give a first step to a computerised management of level crossing.

References

[1] Unified modelling language version1.4. Technical report, OMG, 2001.
[2] Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge University Press, August 1996.
[3] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. METEOR : A successful application of B in a large project. In *Proceedings of FM'99: World Congress on Formal Methods*, pages 369–387.
[4] Philippe Bon, Jean-Louis Boulanger, and Georges Mariano. Semi formal modelling and formal specification: UML & B in simple railway application. In CNAM-Paris, editor, *ICSSEA 2003*, December 2-4 2003.
[5] C. DaSilva, B. Dehbonei, and F. Mejia. Formal specification in the development of industrial applications: The subway speed control mechanism, 1991.
[6] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.
[7] David Harel and al. On the formal semantics of statecharts. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 54–64. IEEE Press, 1987.
[8] L. Jansen and Eckehard Schneider. Traffic control systems case study: Problem description and a note on domain-based software specification. Technical report, Institute of Control and Automation Engineering, Technical UNIVERSITY of Braunschweig, 2000.
[9] C. Morgan. *Deriving programs from specifications*. Prentice Hall International, 1990.