THE ROLE OF FORMAL METHODS IN DEVELOPING A DISTRIBUITED RAILWAY INTERLOCKING SYSTEM

M. Banci[1], A. Fantechi[2], S. Gnesi[1]
[1] *Formal Methods && Tools Group, ISTI - CNR,*
Address: Via G. Moruzzi, 1, Pisa, Italy, I-56124.
Phone: (+39) 050.3152918, Fax: (+39) 050.3152810, e-mail: {m.banci|s.gnesi}@isti.cnr.it
[2] *Università degli Studi di Firenze, Dipartimento di Sistemi e Informatica,*
Address: Via S. Marta, 3, Firenze, Italy, I-50139
Phone: (+39) 055.4796265, Fax: (+39) 055.4796363, e-mail: fantechi@dsi.unifi.it

Abstract: The development of computer controlled Railway Interlocking Systems (RIS) has seen an increasing interest in the use of Formal Methods, due to their ability to precisely specify the logical rules that guarantee the safe establishment of routes for trains through a railway yard. Recently, a trend has emerged about the use of statecharts as a standard formalism to produce precise specifications of RIS. This paper describes an experience in modelling a railway interlocking system using statecharts. Our study has addressed the problem from a "geographic", distributed, point of view: that is, our model is composed by models of single physical entities (points, signals, etc..) that collectively implement the interlocking rules, without any centralized database of rules, which is on the other hand a typical way of implementing such a system (what we call "functional" approach).
We show how a distributed model of this kind may be used to develop a distributed implementation, that employs physically distributed controllers communicating through a "safe" field bus. Ensuring safety of this kind of RIS is entirely based on formal verification.
Keywords: Railway signalling, interlocking, safety critical systems, formal specification, Statecharts, distributed system, field bus

1. INTRODUCTION

A Railway Interlocking System (RIS) is a set of devices and equipments for the safe establishment of routes for trains through a railway interlocking systems due to their advantages in terms of:

- less dependency on operator's failure;
- self diagnostic system checks to improve reliability;
- possibility of global control of wide areas;
- increased capacity in terms of smaller route setting times and greater number of simultaneously live routes.

Typically, a RIS receives a route request; the answer to this request has to go through a series of checks and actions.

The aim of these checks and actions is to guarantee that no train can be driven into a route occupied by another train. Before the computerised interlocking system allows a signal aspect to display green, it checks that the train's route is clear of trains and that all points are locked in the correct positions.

yard. Electronic signalling systems have replaced (and actually are still replacing) old relay-based

The case of Formal Methods in the development of RISs is evident from the list of checks and actions needed to establish the system behaviour, and there is a considerable literature about formalization of interlocking systems (see for example Bacherini *et. al.* 2003; Cimatti *et. al.* 1998; Foschi *et. al.* 2003).

Inside the EuroInterlocking project a trend has developed towards the use of finite state machines for modelling interlocking rules, as Koenig & Einer (2003) described. In particular, a richer form of machines, namely the Statecharts (Harel 1987; Harel *et. al.* 1987), has been considered suitable to express the sequences of checks and actions typical of an interlocking system. Both the UML dialect (OMG 1999) and the Statemate dialect (Harel & Politi 1998) of Statecharts have been considered.

This paper is based on this idea, describing an experience of modelling a railway interlocking system using Statecharts and furthermore implementing the model in a distributed system. Our study has addressed the problem from a "geographic", distributed, point of view, rather than a "functional", centralized, point of view: that is, our model is made up by models of single physical entities (points, signals, etc..) that collectively implement the interlocking rules, without any centralized database of rules and any centralized equipments managing the whole system, which is a typical way of approaching the problem.

The main aim of this study is to investigate the feasibility of the geographic approach for the development of a distributed interlocking system; one of the most interesting issues we are expecting to evaluate, and which has actually motivated our work, is to transpose and implement our geographic model in a distributed system, using devices with a local embedded control program, and communicating among them using a field bus, that is, basing on a technology largely used in automotive and other industrial applications.

Following this approach, formal verification becomes the most important way to guarantee the global safety of the system.

2. GEOGRAPHIC AND FUNCTIONAL MODELLING

A RIS is an embedded system that ensures the safe operation of the devices in a railway station. Such a system controls an arrangement of signals and track points so interconnected that their functions shall be performed in proper sequence and for which interlocking rules are defined in order to guarantee safe operations. A simple (the simplest) example of RIS, that we will use in the paper as a case study, refers to the layout shown in the Figure 1, and is based on a real Italian Interlocking System, as described by Debarbieri *et. al.* 1987.

Interlocking rules are obviously the core of the system, so their correctness is the main objective to be addressed by a formal specification. The rules aim at allowing only the safe combinations of switches positions, signals and trains in a station in order to avoid collisions. The signal indications, handled by the interlocking system, govern the correct use of the routes, authorizing the movement of trains.

The rules usually enforce a predefined sequence of actions: issuing a route request command usually first triggers a check that all the track elements involved in the route are free; in the case, commands are issued for the positioning of points for that route and for locking the track elements. This phase may be followed by a global centralized control over the correct state of the commanded elements, after which the route is locked and signal indications for the route are set.

2.1 A functional approach

Most of computer based interlocking systems use (in their implementation and/or formal specification) some form of centralized data base where the rules of the interlocking logic are stored. The main feature of this approach consists in generating the rules by adopting a design methodology focused on functions, such as the switch points checking function, the routes setting function or the routes verification as illustrated in Fringuelli *et. al.* (1992). This is why we call this approach "functional". These functions are designed basing on a "condition table" (*control table*), which indicates all of the conditions that have to be satisfied before a signal can be switched from red to green to admit a train into the track section beyond (Kolk 1998). The placement of the yard equipments is ignored in this design methodology, it does not exist a direct correlation between the geographic position on the yard of a device and the function that controls it, as implemented in the RIS.

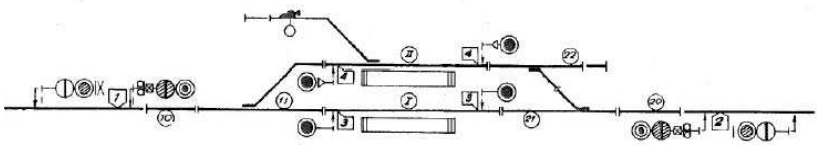

Fig. 1. The simplest railway yard layout.

For this reason it is a hard task to identify the parts of the system stimulated by external events. A specification or implementation following this approach is a melting pot of functionalities not easy to separate. The geographic information of the yard is not available any more after the RIS functional description.

2.2 A geographic approach

A different approach can consider distributing the knowledge of the interlocking rules to objects modelling the geographic placement of physical elements.

An example of geographic specification is the EURIS language (EUropean Railway Interlocking Specification), which is a visual and graphical specification language for railway control systems (Berger *et. al.* 1993; Fokkink & Hollingshead 1998). By using this language it is possible to build in a component-based way the control system from the layout of the station to model. The EURIS specification language, proposed in 1992 and used at Siemens, is able to specify different railway yards using the same generic specification components. Actually it consists of a set of standardized railway control components (Van Dijk *et. al.* 1998), which should be composed together easily, increasing the speed of development and permitting the reuse of components. Each element includes a set of rules inside it, which are able to adjust to different layout configurations. Because EURIS is a graphical language another advantage is that specifications are easy to read, since they immediately represent the physical position of elements in the yard.

The geographic approach reflects directly the placement of the devices on the yard. This point has inspired our proposal for the development of a D-RIS (Distributed Railways Interlocking System); though the specification is done as a centralized piece of work, on which formal verification can be carried on, its geographic structure allows a slicing into independent, distributed, units that can be individually implemented and physically located close to the relevant yard entities.

Distribution of control has already been applied in railway signalling when the movement of trains on a line is concerned: equipment is distributed along the line and connected by fast communication lines, two examples are given in Bacherini *et. al.* (2003) and Haxthausen & Peleska (2000). Distribution of the control of a complex railway yard has not been attempted so far due to the perceived higher safety of a centralized control, based on a single database (the so called "condition table") containing all the rules for a safe movement of the trains within a yard.

The interest in distributing the control of a railway yard may be given by the much lower costs of the interconnections between physical entities, that may be reduced to a single bus and a single power line.

Our proposal aims at establishing a development cycle for RISs in this direction, fulfilling the demanding safety concerns by means of extensive usage of formal verification.

3. STATECHARTS

The Statecharts formalism (Harel 1987) is an extension of the classic formalism of Finite State Machines (FSM), to allow hierarchical parallel interacting state machines to be specified. Transition from a state to another of a single machine (a statechart) is driven by trigger events, which can refer to the state of other machines or to global variables; therefore, the communication activity between statecharts is performed using broadcasting: every event is sent to the whole system, and can be received from any other part of the system. During a transition the actions generate events, (similar to a Mealy machine) which are triggered by conditions on other transitions or on global variables. Chains of internal events generated by only one external event are possible.

The hierarchy feature of Statecharts permits to slice a system into well defined subsystems, so reducing the complexity, and permitting to build a structured model with concurrent parts. The system can be decomposed using AND-states, which evolve singularly and in a parallel way.

Two main dialects of Statecharts are actually used: UML (Unified Modelling Language) State Diagrams (OMG 1999) and Statemate Statecharts (Harel *et. al.* 1990).

3.1 The Statemate tool

In this paper, we follow the style of Statemate Statecharts. The I-Logix Statemate tool (I-Logix 2003; Klose & Damm 2001) supports the editing of the graphical Statecharts notation, but more importantly allows the complete (centralized)

system specification to be executed and graphically simulated, permitting to explore any scenarios determining the system correctness, and evaluating whether the specification meets the requirements. The Statemate simulator allows to execute the model, permitting to verify the behaviour examining the animation of the system and producing test scenarios that may be used in order to test the target system. Furthermore it allows animating panels to have an evidence of the model behaviour, so that we can generate easily "what-if" scenarios. Statemate provides also the automatic generation of a C-based or Ada-based prototyping source code based on the model.

This feature happens to be important for our study, since it permits to generate the code for the distributed devices starting from the global model obtained by joining geographic objects and then slicing it in a set of modules implementable on each controller, physically located to each device.

Another important recent add-on to Statemate is a powerful model checker, which is obviously an advantage in terms of the confidence that can be acquired on the specification correctness. These features of Statemate make it currently superior w.r.t. UML-based tools which allow only editing of State Diagrams.

The Statemate tool supports also another sort of charts, useful to build a structured system: the *Activity Chart*. Activity charts can be viewed as multi-level (hierarchical) data-flow diagrams (DFD). An activity chart describes the functional decomposition of system's capability into functions, or activities, organized into hierarchies. This hierarchy details the functional components, or activities, that the system is capable of carrying out, and how these components communicate through information flow among them. The behaviour of each activity is described using statecharts. In this study the activities are used to represent the distributed devices on the yard and therefore a controlling statechart is associated to each device.

4. STATECHARTS GEOGRAPHIC MODEL SPECIFICATION

We have used a methodology to design an interlocking system starting from its layout and ending in its distributed operational specification. Using this methodology we do not use any sort of global summarizing variables, which is usual instead for a functional approach.

With the term *summarizing variable* we mean a variable whose values depend from the values of a set of other single variables, each related to a physical entity of the layout. As an example we can consider a variable associated to a route, that is true if and only if at least one of the variables recording the occupancy of the track circuits belonging to the route is set to true. The use of summarizing variables, though useful for abstracting certain global aspects of the system, makes the model more distant from the physical topology, and so it is less interesting for our approach. Note that the variable associated to a route of a functional model corresponds to a route reservation relay in a relay-based RIS. In our proposal, we therefore loose the uniqueness of such information in the system, since it is replicated and distributed over several objects.

The experience discussed in this paper has been actually preceded by a modelling by Statemate statecharts of the same interlocking system, using a classical functional approach: we started by a condition table, which defines the conditions that have to be respected. Specific modules were dedicated to record commanded routes; these modules had the responsibility of checking whether the interested track circuits were free; as is evident from the context, those modules had only logical function and had no correlation with physical devices.

On the contrary, in the geographic approach it is a module dedicated to the management of each track circuit that has the responsibility to check its compatibility with commanded routes and with other events happening in the system. In the same way, all the activities performed by functional objects have been distributed to these geographic objects. There is not a single core of the RIS, but each device has its own logics that checks every command coming from the other devices and elaborating them performs the correct action independent from the decision of the other objects. Hence, the control for example of the correct position of a particular switch point is performed from all the interested elements to that position and not by a single object dedicated to the management of all the switch points.

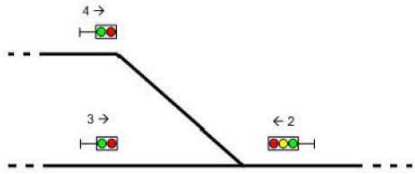

Fig. 2. An example of routing.

Each object of the model implements the rules that interest only that object. If, for example, we consider the three semaphore of Figure 2, the model will include an object for each semaphore. The object related to the semaphore 4, which permits the movement of a train on the right direction, should control that the red lights of semaphores *3 right* and *2 left* are fired and also that the green lights are switched off. This control is done not looking at a global summarizing variable, which in the example would show that the route is free, but communicating with the objects that control the other semaphores and devices.

In this way, we obtain a model whose structure reflects the layout of the railway yard so that we can easily implement it in a distributed system. This has positive effects on the readability of the model and on its reliability, because of the redundant checks replicated on many different devices independent among them.

On the other hand, we loose on generality: in the functional approach the module handling all switch points can be generic, and it is the condition table that embeds the knowledge about the specific rules for the railway yard. The objects that we have designed have not a generic behaviour usable in any different geographic layout (like EURIS): we have to redesign them for any different station, though following expected patterns with predetermined rules. The generalization was not our main concern, which has been left for future work.

The consequences of this geographic approach are:

- The structure of the model reflects the geographic topology of the yard.
- The elements of the model replicate the behaviour of the physical components of the yard.
- The elements of the model embed all the logical rules interesting the corresponding physical components (in order to avoid the usage of summarizing variables).
- The model will be able to be translated directly to a physically distributed implementation.

In a previous work presented at FMICS '04 workshop (Banci & Fantechi 2004) we have analyzed the benefits of the geographic approach in the modelling of a RIS from the point of view of regression test case generation in case of changes to the topology of the yard.

The last consequence of the above list suggests instead to push forward an innovative philosophy in the implementation of a RIS, that bases on the success distributed controllers are enjoying in other safety critical fields, such as automotive. This is what constitute the proposal which we present in this paper. In particular, we go on in this chapter by detailing the geographic Statemate model of the RIS. This model is still a centralised piece of work, that can be simulated and validated on a host machine.

Deployment over distributed targets will be described in section 5. Safety concerns are addressed both in section 5 and in section 6, in which the whole proposed development cycle is discussed.

4.1 Structure of the geographic layered model

The geographic model is built following a layered abstraction, which consists of three layers: *Command* (human) layer, *Logical* layer and *Physical* layer (Figure 3). The first layer (Command) is dedicated to the interaction with operators or other systems, which send commands to the RIS.

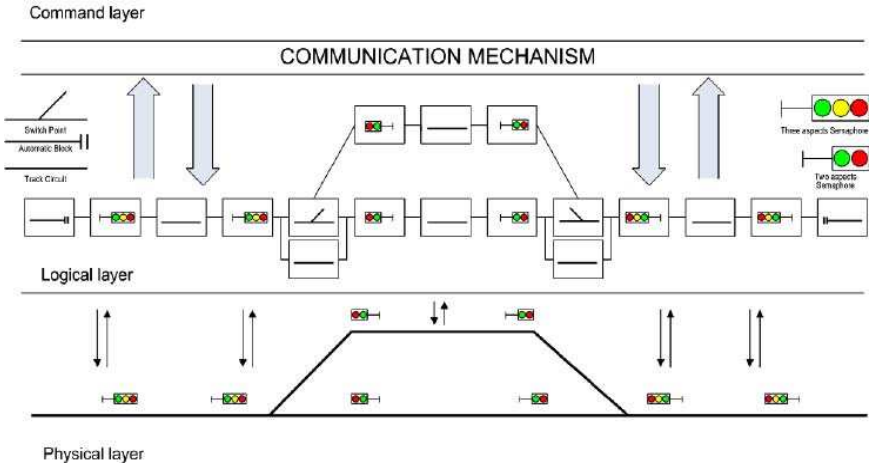

Fig. 3. Illustration of the layered interlocking architecture.

At the lowest level (Physical), there are the yard devices and equipments, which have to be commanded and controlled by the RIS and in which the single FSM controlling them will be implemented. This level is constituted of the actual device interfaces, with actual variables used to control the yard. The middle level is the core of the RIS, where the interlocking rules are specified in a centralized way. It is formed by a separate object for each physical device. Figure 4 shows how the objects are interconnected with the command and the physical layer. The state of any object is one to one related with the actual state of physical device, and for this reason it will be able to be implemented directly on device controllers. Every object related to a particular route is able to receive the command requesting that route, in which case it performs suitable controls toward the physical layer and the other objects of the logical layer. When a route reservation command is sent by an external system (also human), this message is sent to all the objects related to that route. Then all the objects evaluate their rules interacting each other, passing values of variables, to confirm the received command.

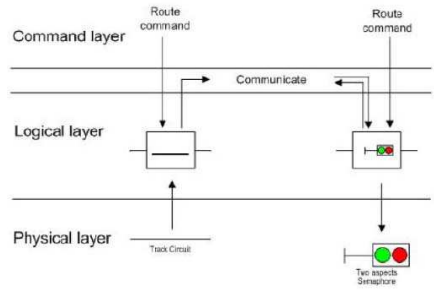

Fig. 4. Illustration of inter-object communication.

Inside each object it is therefore inserted only a slice of the logic, that is usually centralized in a classical functional approach. Every object is all the time active and elaborates the information coming from other objects. In the model does not exist any coordinator object, but they are all to the same level and they embed the logic permitting them to coordinate among themselves: e.g. if a route command was sent, each object interested by this route would perform a control action on the other objects: this action is done by all the objects in parallel and independently.

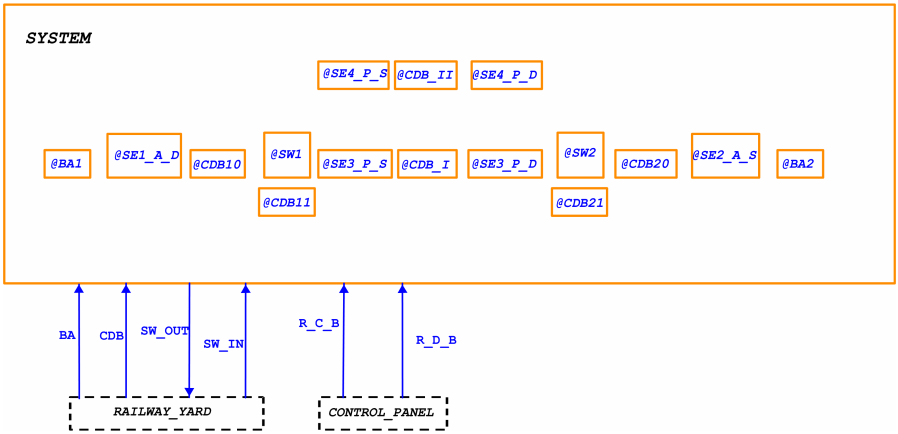

Fig. 5. The first level of the statechart model and the distribution of the internal activity charts.

4.2 The statecharts model specification

The system is specified combining the geographic elements as it is illustrated in Figure 3 and 4. Each geographic element is defined by an activity chart specified using the Statemate statechart formalism. As shown in Figure 5, the main level of the model consists of an activity chart, composed by several activities, which are strictly related to physical objects placed on the yard. At a lower level each block is formed with a set of nested subactivities and statecharts that implement the interlocking rules of single devices. We can note that the topology of this level is exactly corresponding to the geographic layout of the yard (refer to Figure 1). As we said previously, statecharts interact among them using shared variables. Shared variables are therefore used to implement the communication between objects: every block checks which is the state of other nearby objects *snooping* some of their state variables.

An example of the statechart describing the behaviour of an activity is shown in Figure 6 where a track circuit manager is illustrated: the figure shows the use of a logical state such as that used to reserve the object. These local states are needed because we do not have any global object that records which elements are in use, so the control logic has been decentralized. We can note that the chart communicates with plenty of other objects, such as semaphores and other distinct track circuits, by looking at shared variables. Indeed, interlocking rules are distributed over the conditions for the transitions in each activity chart.

Figure 7 represents the statechart controlling a green light: when an operator (either human or system) gives a command, this statechart and all the other statecharts controls that the track circuits related to it are reserved and the track circuits incompatible with it are free. Though each chart works in parallel with the others, they are strictly interrelated by this massive usage of shared variables.

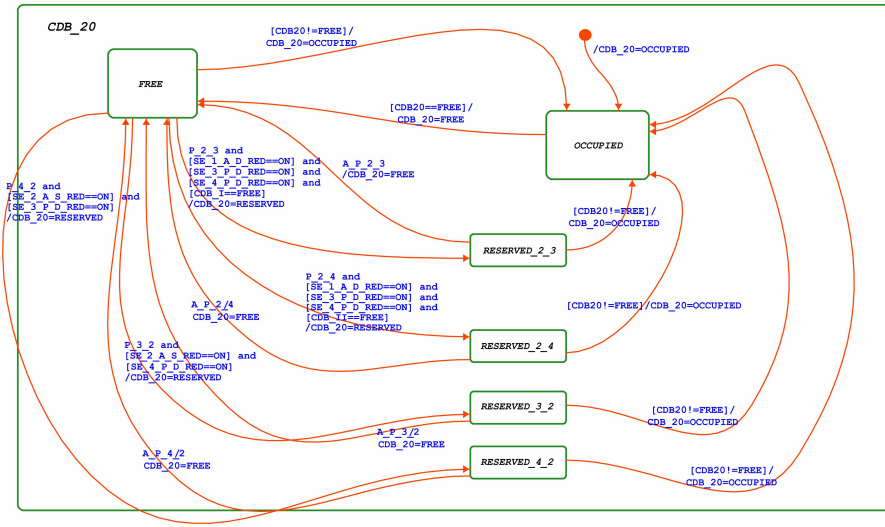

Fig. 6. A track circuit statechart.

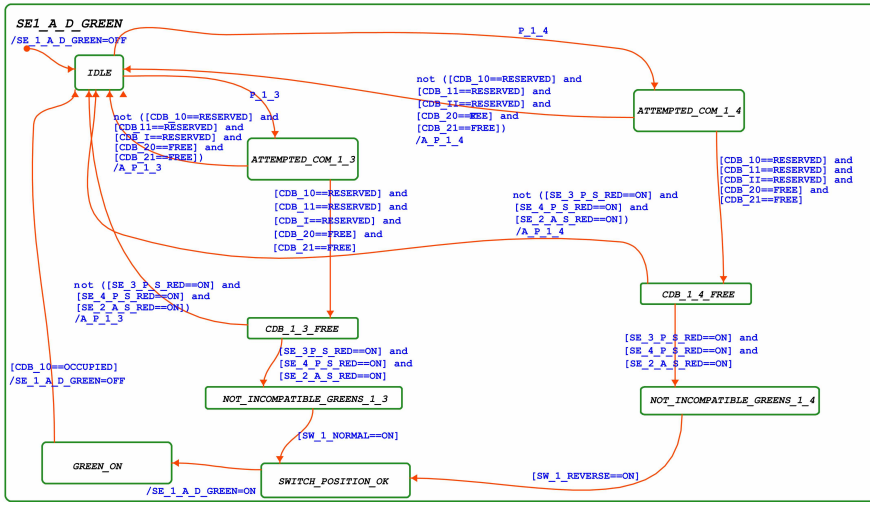

Fig. 7. The statechart controlling the green light of a semaphore.

5. THE DISTRIBUTED SYSTEM ARCHITECTURE

We have observed that the geographic approach used to model the RIS keeps the original topology of the system, and this fact has inspired our proposal to physically distribute the control by deploying each activity in a separate controller physically close to the controlled entity. The distribution consists in generating slices of the geographic model as is illustrated by the Figure 8. An obstacle to physical code distribution is however represented by the shared variables used in the model for communication between the separate activities. The semantics of Statecharts requires that every activity is able to read and write shared variables at any step. In a distributed implementation, variables need either to be associated to an activity, which should provide for safe reading and writing by other activities, or to be replicated among the interested activities, and in this case consistency of the replicas should be guaranteed.

The synchronous nature of the operation of Statecharts considers variables values to be read at the beginning of a step. Only when the evaluation of variables has dictated the live transitions that can be fired, one of them is fired and the associated actions, including writing on variables, are performed. It is possible to perform automatically checks that guarantee that no conflict is raised about simultaneously writing of a variable by two activities, in order to avoid race conditions.

This operational semantics allows to consider a distributed implementation based on the adoption of a field bus, by which variable values are broadcasted at the beginning of a new operation step, and by which writing commands issued by (one and only one) activity are conveyed to the owner of the variable at the end of the step.

Indeed, our idea is based on the rapid development of safe field bus area: we think that the market is now mature to accept this kind of approach in the railway area as well, given the large number of applications of field buses in different safety-concerned industrial areas: from factory automation to *fly-by-wire* and *drive-by-wire*, in avionics and automotive areas respectively.

Due to the synchronous operations typical of Statecharts, a good candidate to act as the basic platform, on which our approach is based, is the architecture named TTA (Time Triggered Architecture) as Kopetz & Bauer 2003 described.

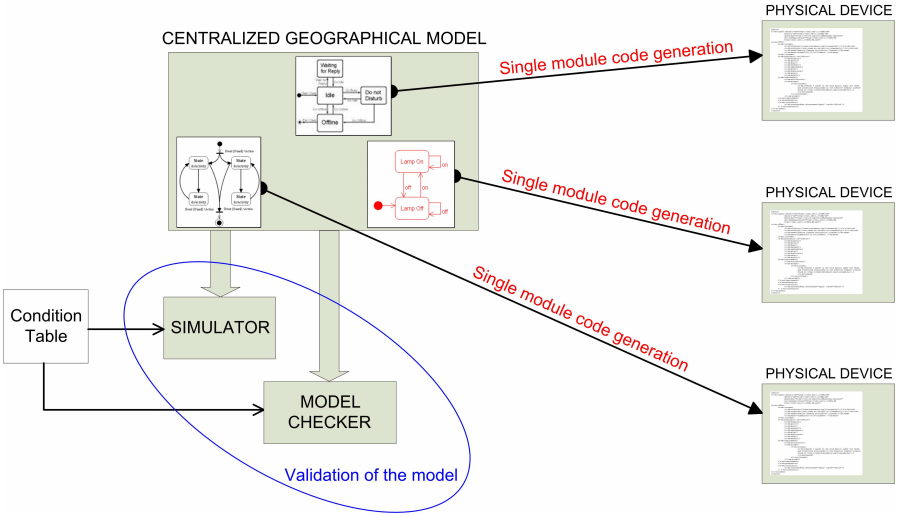

Fig. 8. Development cycle

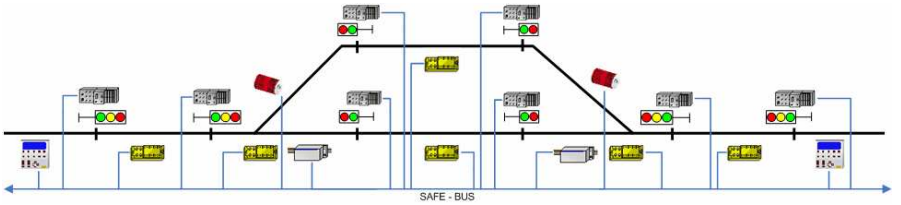

Fig. 9. The network devices deployment.

This architecture has been created for the implementation of dependable distributed embedded systems, and permits to decompose a large real-time application into nodes: obviously the main critical feature consists of the communication mechanism and the synchronization one. In the TTA, the system maintains a fault tolerant global time at every node. This global time permits to reduce the communication complexity allowing the use of shared variables for communication purposes; events that happen in the distributed system at different nodes at the same global clock-tick have to be considered simultaneous. The TTP (Time Triggered Protocol) is in charge of guaranteeing the consistency of different views of the same variable at any given clock tick.

Another issue that should be taken into account is given by the safety requirements in case of a fault. In our proposals, faults can occur in any distributed controller. The basic safety requirements, to be achieved both by exploiting the fault tolerant features of the bus protocol, and by properly designing the distributed components, are that:

- any failure of a component is reduced to a crash of the component itself, so that a fail-silent policy is enforced;
- the presence of a silent component does not undermine the safe operation of the interlocking; that is, no route for which the failed component is needed can be set and acknowledged.

For increased availability, we can add the requirement:

- any failed (fail-silent) component does not affect the correct setting of a route which is totally (geographically) independent from the component interested by the failure.

Note that the distributed system is formed from redundant controls, which are located at every device (Figure 9). The redundancy of the controls exhibited by the geographic approach can be considered as a positive safety measure: a decision about the establishing of a route is taken only if all the controls have been successful; the controls are redundant, but diverse and independent, hence they constitute a safeguard against software faults. Note that this safety measure is not due to the exploitation of the TTA architecture (or of any other suitably fault tolerant bus), but is intrinsic in the model.

The system architecture is completed by a monitoring computer (or more than one computers) attached to the bus, able to read the variables values that are exchanged on the field bus, which uses such data for diagnostics, for displaying to the humans the state of the yard and of the interlocking system, and for logging data about the system. The monitoring computer can also be used as an added safety measures by taking in charge the forcing of the system in a safe state in case it detects anomalous variable values.

6. DEVELOPMENT CYCLE

What is needed to implement a distributed RIS can be summarized in the following development cycle, which includes validation activities as well:

1. *Condition table:*

This table is taken as the contractual input for the process, and fully describes the interlocking rules according to the given yard topology.

2. *Statemate design using geographic approach:*

As described in section 4, a geographic model using activity and statecharts is developed.

3. *Validation of distributed design:*

The geographic model is validated by means of two different alternative methods (which actually should be both applied for increased safety):

3.1 tests played by simulation:

We are able to simulate the whole model with the Statemate simulator tool, which permits to

interact directly by using a panel appropriately created as well (Figure 10).

Extensive tests should be carried on defining suitable test scenarios, on the basis of the information given by the condition table.

3.2 Condition table based safety properties proved by MC:

Safety properties described by the condition table should be defined so that they cover the overall safety requirements. Typically, properties of the kind: "two conflicting routes can never be set simultaneously" should be expressed and verified of the model by a model checker (typically, the Statemate model checker). Model checking is able to guarantee that such properties are always satisfied by the model, while simulation may leave some dangerous execution paths unexploited.

4. Fault injection:

Extensive verification, again by simulation and/or model checking, should be done in order to validate the fail-safe behaviour of the model. Typically, faults should be injected in the model (e.g. by forcing a fail silent behaviour of some objects) in order to test the overall safety of the system in presence of faults.

5. Automatic code generation

5.1 Statemate code generation:

From the statechart geographic centralized model it is also possible to generate C or ADA code by using the automatic code generator tool, which is part of the Statemate tool, for every single device. Because of the detailed nature of the model, the code generated is immediately usable without need of any other translation into lower level languages, except for the communication interface which is not part of the requirement specification. The resulting code shares with the geographic model the correspondence between software modules and yard devices. For this reason there is the possibility to generate code for each module, to be deployed on a local controller.

5.2 Shared variables implemented through field bus protocol:

The variables which are shared between the obtained software components should instead be implemented basing on the safe protocol established to this purpose over the adopted field bus.

6. Physical deployment and integration:

Deployment of the various modules over the distributed controllers connected to the field bus is now possible.

7. System in field testing:

Though the extensive validation effort carried on the model and the automatic generation of code is enough to guarantee the safety of the system, in field testing is necessary to guarantee that any possible interference from the physical world does not undermine the safety and functionality of the system.

In our experiments we have addressed parts 1, 2, 3.1 above, and we are currently working on issues 3.2 and 4; we also are investigating the current industrial field bus state of the art, in order to complete our study of the feasibility of our proposal.

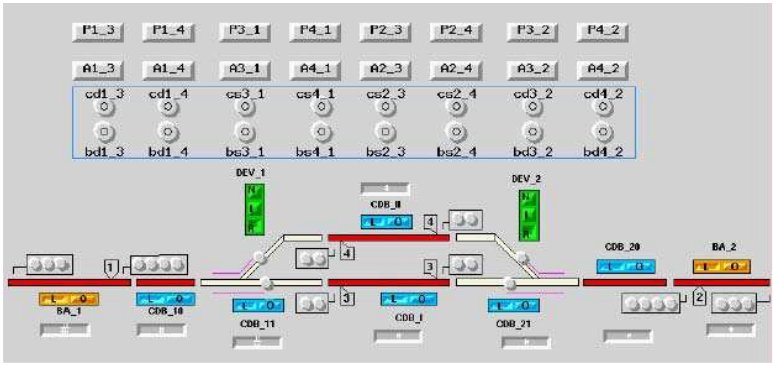

Fig. 10. The simulation control panel.

7. CONCLUSIONS

The experience we have presented is part of a wider research project aiming at investigating the design of RIS by means of different Statechart dialects and different commercial tools that support Statecharts, such as for example Stateflow (MathWorks), Telelogic TAU Generation 2 (Telelogic), RealTime Studio (Artisan Software), visualSTATE (IAR Systems). In this paper we have pushed a geographic approach to the modelling of RISs to the point that the interlocking logic can be entirely distributed on "in the field" local controllers, following a trend consolidated in automotive and avionics applications, based on the use of robust field-bus. We have shown that in this approach formal verification gets the role of the primary method to asses the safety of the system.

REFERENCES

S. Bacherini, S. Bianchi, L. Capecchi, C. Becheri, A. Felleca, A. Fantechi, E. Spinicci (2003).
Modelling a Railway Signalling System using SDL, *Proceedings of the International Symposium on Formal Methods for Railway Operation and Control Systems (FORMS 2003)*, Budapest, Hungary, 15-16 May 2003.

M. Banci, A. Fantechi (2004).
Geographical vs. Functional modelling by Statecharts of Interlocking Systems, *Proceeding of the Ninth International Workshop on Formal Methods for Industrial Critical Systems (FMICS 04)*, Linz, Austria, 20-21 September 2004.

J. Berger, P. Middelraad, and A.J. Smith (1993). EURIS, The EUropean Railway Interlocking Specification. UIC, Commission 7A/16, 1992. In *IRSE Proceedings* 1992/93, pages 70-82, 1993.

A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli and P. Traverso (1998). Formal Verification of a Railway Interlocking System using Model Checking, *Formal Aspects of Computing*, Vol 10, 361-380.

P. E. Debarbieri, F. Valdambrini, E. Antonelli (1987).
A.C.E.I. Telecomandati per linee a semplice binario, schemi I0/19. *CIFI Collana di testi per la preparazione agli esami di abilitazione*, Quaderno 12, 1987.

W. J. Fokkink, P. R. Hollingshead, (1998). Verification of Interlockings: from Control Tables to Ladder Logic Diagrams, Proceedings *of the 3rd Workshop on Formal Methods for Industrial Critical Systems -FMICS '98*.

U. Foschi, M. Giuliani, A. Morzenti, M. Pradella, P. San Pietro (2003).
The role of formal methods in software procurement for the railway transportation industry, *Symposium on Formal Methods for Railway Operation and Control Systems (FORMS 2003)*, Budapest, Hungary, 15-16 May 2003.

B. Fringuelli, E. Lamma, P. Mello, G. Santocchia (1992).
Knowledge Based Technology for Controlling Railway Stations. *In IEEE Intelligent Systems*, Volume: 7, Issue: 6, Dec. 1992, Pages: 45-52.

D. Harel (1987).
Statecharts: A Visual Formalism for Complex Systems. *Sci. Comput. Programming 8* (1987), 231-274.

D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull Trauring, M. Trakhtenbrot, (1990)
STATEMATE: A Working Environment for the Development of Complex Reactive Systems, *IEEE Transactions on Software Engineering*, Vol. 16, N. 4, April 1990, pp. 403-414.

D. Harel, A. Pnueli, J. Schmidt and R. Sherman (1987).
On the Formal Semantics of Statecharts, *Proc. 2nd IEEE Symp. on Logic in Computer Science*, Ithaca, NY, 1987, pp. 5464.

D. Harel and M. Politi (1998).
Modelling Reactive Systems with Statecharts: The STATEMATE Approach, McGraw Hill, 1998. (Early version titled: The Languages of STATEMATE, I-Logix, Inc., Andover, MA, 1991.)

A. E. Haxthausen, J. Peleska (2000)
Formal Development and Verification of a Distributed Railway Control System, *IEEE Transactions on Software Engineering*, Vol. 26, No. 8, pp. 687-701.

J. Klose, W. Damm, (2001).
Verification of a Radio Based Signalling System Using the STATEMATE Verification Environment , *Formal Methods in System Design*, 19(2).

N. H. Köenig, S. Einer (2003).
The EuroInterlocking Formalized Functional Requirements Approach (EIFFRA), *Symposium on Formal Methods for Railway Operation and Control Systems (FORMS 2003)*, Budapest, Hungary, 15-16 May 2003.

G. Kolk (1998).
Formal methods: Possibilities and difficulties in a railway environment from a user perspective. In *Proceedings of the Third International Workshop on Formal Methods for Industrial Critical Systems*, May 25-26, 1998.

H. Kopetz, G. Bauer (2003).
The Time-Triggered Architecture, *Proceeding of the IEEE Special Issue on Modelling and Design of Embedded Software*. Vol. 91, Issue: 1 Jan 2003, 112-126.

Object Management Group, (1999).
Unified Modelling Language Specification, Version 1.5, *http://www.omg.org/technology/documents/formal/uml.htm*.

Statemate Magnum (2003)
Simulation Reference Manual. I-Logix Inc. Burlington, MA USA, 2003.

F. Van Dijk, W. Fokkink, G. Kolk, P. Van de Ven and B. Van Vlijmen, (1998).
EURIS, a Specification Method for Distributed Interlockings, *Proc. SAFECOMP '98, Heidelberg, Germany, October 5-7, 1998, in Lecture Notes in Computer Science*, vol. 1516, Springer.