Modeling a distributed railway interlocking system with object-oriented Petri nets

X. Hei[1], H. Mochizuki[1], S. Takahashi[1], H. Nakamura[1], M. Fukuda[2], K. Iwata[2] & K. Sato[2]
[1]*College of Science and Technology, NIHON University, Japan*
[2]*Railway Technical Research Institute, Japan*

Abstract

Great progress of distributed technology and intelligent terminals makes it possible to develop a distributed railway interlocking system (DRIS). In this paper, a modelling method of DRIS is presented by using G-nets, which are Petri nets extended with object-oriented concepts. The modelling method improves maintenance and reusability remarkably. Based on the models, the DRIS can be implemented with an object-oriented language such as C++ or JAVA.
Keywords: distributed technology, railway interlocking system, G-nets, Petri net.

1 Introduction

Computerized interlocking system has been developed for many years [1]. Conventional interlocking system, which is named electronic interlocking system, is composed of central control computer, electrical control part and mechanical part. These systems feature that cost is high and updating of terminal devices is inconvenient.

On the other hands, study and application of distributed control technology and field-control technology have become a new stage [2, 3]. Especially, with the development of ubiquitous technology [4], intelligent terminals or intelligent devices have been possible. All these progress make it possible developing distributed railway interlocking system (noted as DRIS).

In distributed interlocking system, all devices are intelligent, and they communicate with each other directly without centralized interlocking computer, as shown in figure 1. All these intelligent devices have functionality units which are shown in figure 2. The logics of DRIS also become different from centralized interlocking system.

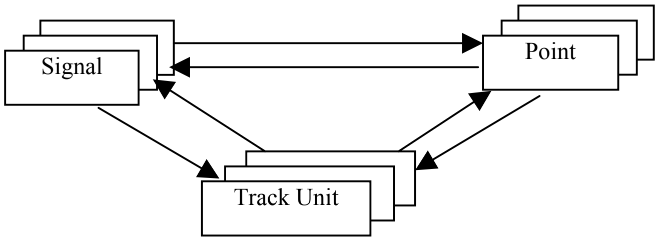

Figure 1: Distributed railway interlocking system (DRIS).

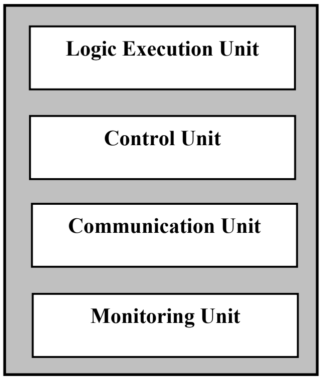

Figure 2: Components of intelligent devices.

DRIS is an autonomous decentralized and distributed real–time control system. The complexity of such system requests new system modeling techniques to support reliable, maintainable and extensible design.

At the same time, as the most important property of railway interlocking system, fail-safe has to be assured. So the correctness verification of new logic also needs a new modeling technique. Further, the dependability evaluation of DRIS is based on a proper DRIS model, which can describe the DRIS model precisely. Some efforts have been made for modeling the railway interlocking system [6, 7]. But they did not either consider DRIS or establish a uniform and hierarchical model.

In this paper, we proposed a hierarchical modeling method for DRIS with G-nets [5] which is Petri net extended with object-oriented concepts.

The reason why we model DRIS based on G-nets is that our analysis of DRIS is also based on object or class. Each device can be thought as an object, namely an independent module. They have their own functions, and they communicate with each other via some well-defined interfaces.

G-net is a modeling tool which is a Petri net extended with Object-oriented concepts, namely, the object, with which the model of DRIS can be simplified evidently. An object packs many internal details of realization. To some extend, G-nets can be thought as a high-level Petri net.

The analysis and evaluation of G-nets model can be converted to analysis and evaluation of Petri nets, which have many available analysis methods and tools.

The paper is organized as follows: section 2 gives an overview of G-nets; modeling approach of DRIS is presented in section 3, and a simple example of station layout and its analysis are presented in section 4; finally, we conclude this paper in section 6.

2 G-nets

G-net is a Petri net based on multi-level executable specification model which incorporates the concepts of module and system structure [5, 6]. A G-nets system comprises of a number of G-nets, each of which represents an independent module. These modules communicate with each other through well-defined interfaces, that is, the methods of G-nets.

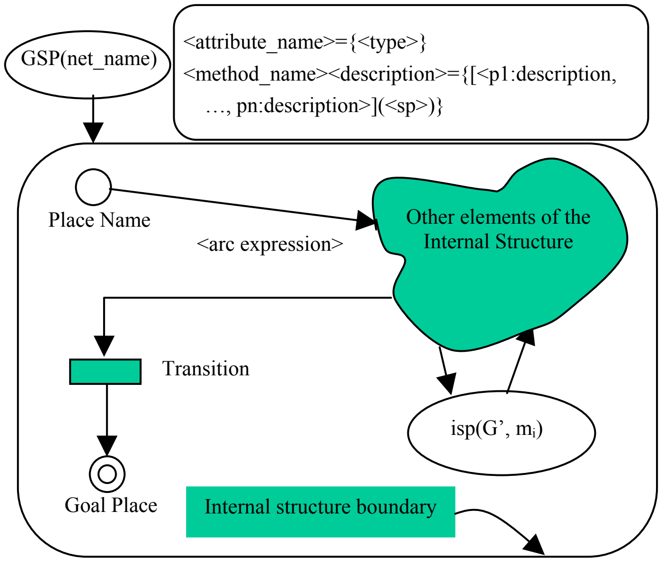
GSP(net_name)
<attribute_name>={<type>}
<method_name><description>={[<p1:description, …, pn:description>](<sp>)}
Place Name
<arc expression>
Other elements of the Internal Structure
Transition
isp(G', m_i)
Goal Place
Internal structure boundary

Figure 3: Notations of a G-net.

A G-net is composed of two parts: a special place called *Generic Switch Place (GSP)* and an *Internal Structure (IS)*. The *GSP* provides the abstraction of the module, and serves as the only interface between the G-net and other modules. The *IS*, a modified Petri net, represents the detailed internal design and realization of the module. The notation for G-nets is shown in figure 3.

The internal structure of the net(*IS*) is enclosed by a round corner rectangle. The *GSP* is indicated by the ellipse in the left upper corner of the *IS* boundary. The inscription *GSP(net_name)* defines the name of the net. The rounded corner rectangle in the upper right corner of the *IS* boundary is used to identify the methods and the attributes for the net, where: *<attribute_name>={<type>}* defines the attributes for the net, *< attribute_name >* is the name of the attributes, and *<type>* is a type for the attribute, restricted to the set of non negative integers; *<method_name>* and *<description>* are the name and description for the method respectively. *<p1:description,…,pn:description>* is a list of arguments for the method, and *(sp)* is the name of the initial place for the method. An ellipse in the internal structure represents an *instantiated switch place(isp)*, which is used to provide inter-G-Net communication, while a circle represents a normal place. The inscription *isp(G',*m_i*)* indicates the invocation of net G' with method m_i. Executing the *isp* primitive implies invoking G'(by sending a token to G') based on the specified methods. This token contains the parameters needed to define the tokens for the initial marking of the invoked net. A rectangle represents a transition that may have an inscription associated with it. This inscription may be either an attribute or a firing restriction. A double circle represents the termination place or goal place. Places and transitions are connected through arcs that may carry an expression.

In the internal structure, places represent primitives, while transitions, together with arcs, represent connections or relations among the primitives. A set of special places called *Goal Place* represents the final state of the execution, and the results (if any) to be returned.

From the description above, we can see that a G-net model essentially represents a module or an object. A G-nets system supports incremental design and successive modification. We find it suitable to model DRIS.

3 Modeling interlocking system

3.1 Function modules of intelligent devices

In DRIS, there are three kinds of intelligent devices: signal, point and track unit. When a train is coming, a route will be set, related signals have to be locked in some status, track units must be in some status and related points must be locked in requested position. When all conditions of route setting are satisfied, the start signal or home signal of the route changes to permission status, which indicates the coming train can go on.

Corresponding with the components shown in figure 2, an intelligent device of DRIS is composed of 4 functionality modules, which are listed in figure 4, namely, the communication module, computing module, control module and monitoring module. Communication module allows devices to communicate with other intelligent devices or other system, such as ATS. Computing module functions as logic computing and judgment, which decides whether the logic status is correct or not, what status it should be, etc. Control module is in charge of controlling actions of devices, such as changing status, locking or unlocking. The monitoring module monitors the field device status by timely polling the device and alarms when there is abnormal status.

These modules are independent in structure and functionality, and they coordinate with each other via some interfaces.

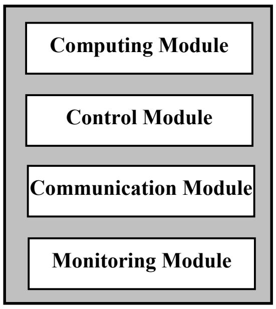

Figure 4: Modules of intelligent devices.

3.2 Modeling DRIS with G-nets

Because the communication, computing and control module of interlocking devices are similar to each other, we introduce only these modules of signal and the monitoring module of track unit. Figure 5 is G-net model of communication module of signal. There are 3 methods: *rm*(receive message), *sm*(send message) and *rr*(route request). The communication module of point and track unit has no the *rr*(route request) method. In method *sm* and *rm,* the parameters are *id* and *data*. *Id* represents network id of destination devices; and *data* represents communication data, which maybe comprises of logic status information. Parameter of method *rr* is *rid*, which indicates the id of requested route.

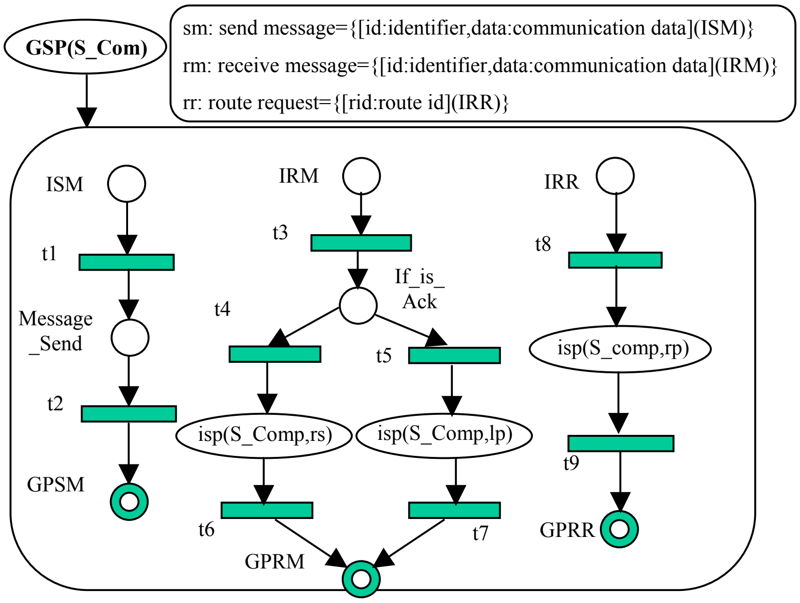
GSP(S_Com)
sm: send message={[id:identifier,data:communication data](ISM)}
rm: receive message={[id:identifier,data:communication data](IRM)}
rr: route request={[rid:route id](IRR)}
ISM
t1
Message _Send
t2
GPSM
IRM
t3
If_is_ Ack
t4
t5
isp(S_Comp,rs)
isp(S_Comp,lp)
t6
t7
GPRM
IRR
t8
isp(S_comp,rp)
t9
GPRR

Figure 5: G-net representing communication module of signal.

In figure 5, *Message_Send* and *If_is_ack* are primitives. They can be executed without calling other methods. The return value of primitive *If_is_ack* is used to judge whether the received message is an acknowledgment from other devices, i.e. whether it is a response message or not. If it is an acknowledgment, then call *rs*(means route setting) method of module *S_Comp*, which changes the signal to permission status when all acknowledgments from route-related devices are received. Otherwise, the received message must be an order that indicates what status of signal should be. In this case, method *lp*(means logic process) of module *S_Comp* is called.

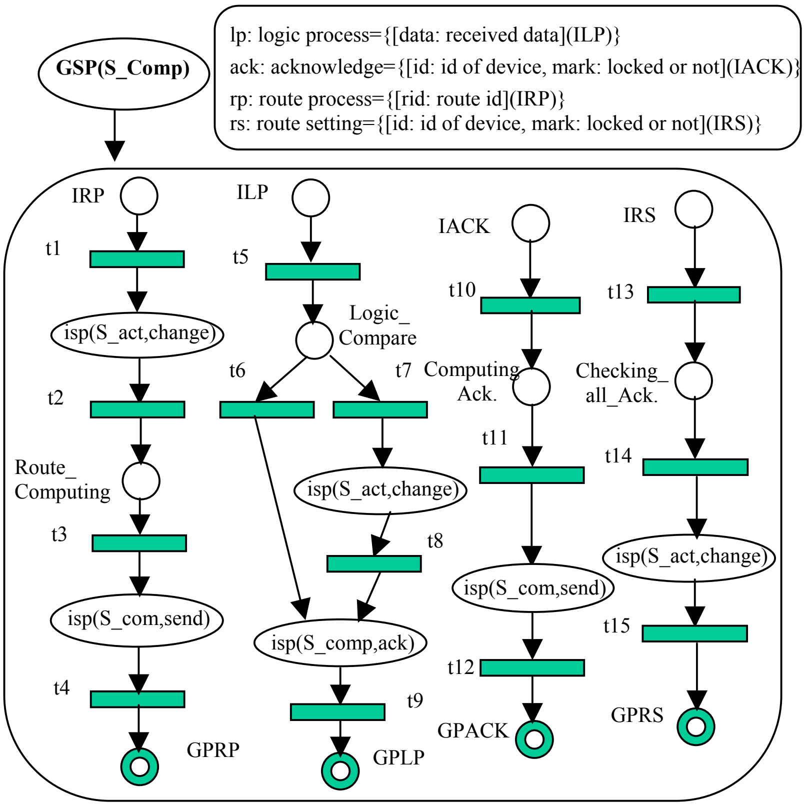

Figure 6: G-net representing computing module of signal.

Figure 6 is G-net model of computing module. There are 4 methods in module *S_Comp*. When signal receives route request via method *rr* of module *S_Com*, method *rp* changes its status to *route setting*, and computes the status information of other route-related devices, then send the result to these devices by *isp(S_com,send)*, which call *send* method of module *S_Com*. Method *lp* executes logic processing. There are 2 circumstances that are considered when receiving message, one is the logic status coming from network is the same as current logic status (transition t6 is fired); the other one is different. In the second case, it is necessary to change logic status and lock it. Finally, devices acknowledge a message by invoking method *ack* of module *S_Comp* to the signal that starts *route setting*. The *rs* method starts when an acknowledgment is received from other devices. When all responses are received and they satisfy the requirements (primitive *Checking_all_Ack* checks the acknowledgment message), signal change itself to *permission* (display green).

The computing module of point and track unit is different from that of signal. They have no method *rp*(route process) and *rs*(route setting).

Figure 7 is G-net model of the control module which is named *S_Act*. In module *S_Act*, there are 3 methods: *change*, *lock* and *unlock*. Method *change* changes status of device; *lock* locks the logic status of device, such as locking point on reverse or normal; *unlock* carries out unlocking action of device.

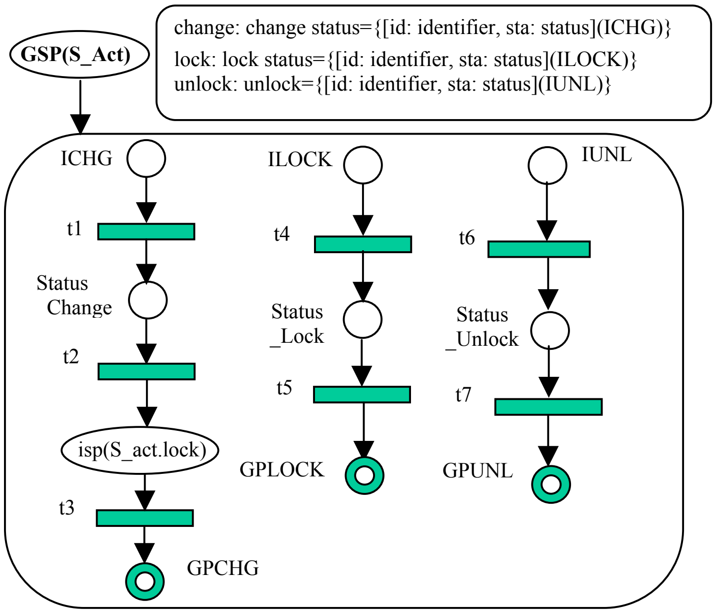
change: change status={[id: identifier, sta: status](ICHG)}
lock: lock status={[id: identifier, sta: status](ILOCK)}
unlock: unlock={[id: identifier, sta: status](IUNL)}
GSP(S_Act)
ICHG
t1
Status Change
t2
isp(S_act.lock)
t3
GPCHG
ILOCK
t4
Status _Lock
t5
GPLOCK
IUNL
t6
Status _Unlock
t7
GPUNL

Figure 7: G-net representing control module of signal.

Figure 8 shows the monitor module of track unit. There are 3 methods in the monitor module, namely, *CheckTrain*, *NotifySignal* and *CheckPoint* methods. *CheckTrain* method checks the status of train(whether the train is on the track), *NotifySignal* method sends train status to signal, and *CheckPoint* inspects the status and position of point which is on some tracks.

These modules show the inter-module interface as well as internal realization of the module. By introducing G-nets model with data domain, the system can be represented evidently. The models can be used to develop DRIS system with any object-oriented programming language, such as C++ or JAVA, etc.

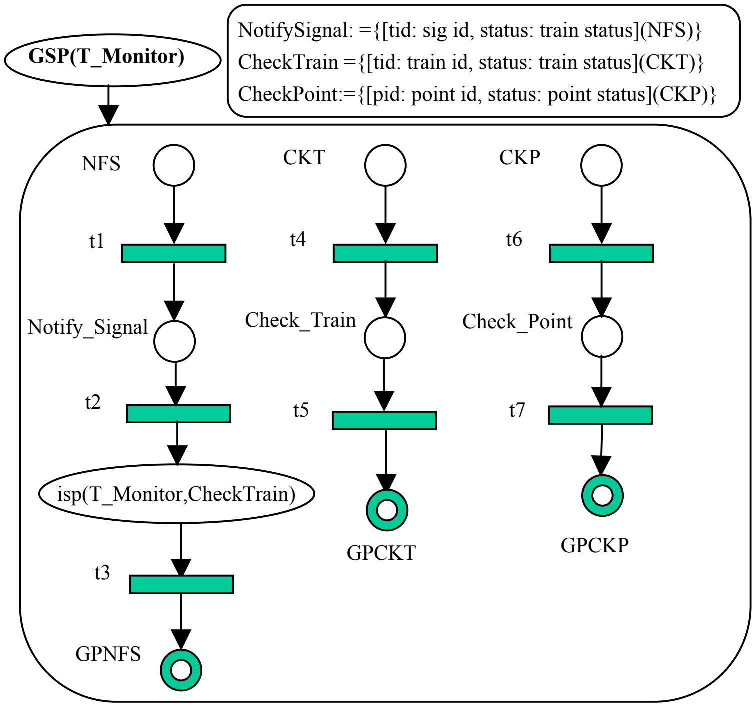
NotifySignal: ={[tid: sig id, status: train status](NFS)}
CheckTrain ={[tid: train id, status: train status](CKT)}
CheckPoint:={[pid: point id, status: point status](CKP)}
GSP(T_Monitor)
NFS
CKT
CKP
t1
t4
t6
Notify_Signal
Check_Train
Check_Point
t2
t5
t7
isp(T_Monitor,CheckTrain)
GPCKT
GPCKP
t3
GPNFS

Figure 8: G-net representing monitor module of track unit.

3.3 The model architecture of DRIS

The primitives, which are shown as a circle in the G-nets model of DRIS, can be analyzed further. Thus a hierarchical Petri nets model with several levels can be established until every possible state is represented.

With G-nets, the DRIS can be analyzed at any level, which makes the DRIS G-net model to become a uniform model.

4 Analysis of a DRIS example

Figure 9 is an example of DRIS, which is used to describe the modeling approach. There are 8 routes, 4 home signals, 4 start signals, 2 points and 2 platforms. Signal has 3 statuses: permission, refusal, and route setting. Table 1 is interlocking information table which is related with route X→1.From figure 9, we find that signal S1 is the home signal of route X→1, and when X→1 is requested, signal S3 must be locked in refusal status.

As a simple illustration, figure 10 shows the invocation process of signal S1 and signal S3 when route X→1 is requested. When X→1 is requested, the route request is sent to signal S1. S1 changes its status to route setting, then searches route related devices and computes what logic status the devices should be. When the process is completed, signal S1 send the computing result to network.

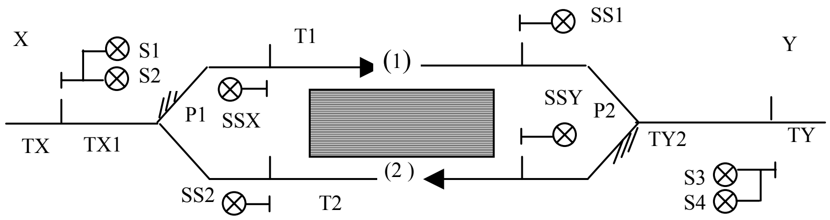

Figure 9: Layout of interlocking example system.

Table 1: Interlocking information related with route X→1 in figure 9.

Route	Signal display state	Point state	Related track unit
X-1	S1-G, S3-R, SS2-R	P1-N, P2-R	TX, TX1, T1

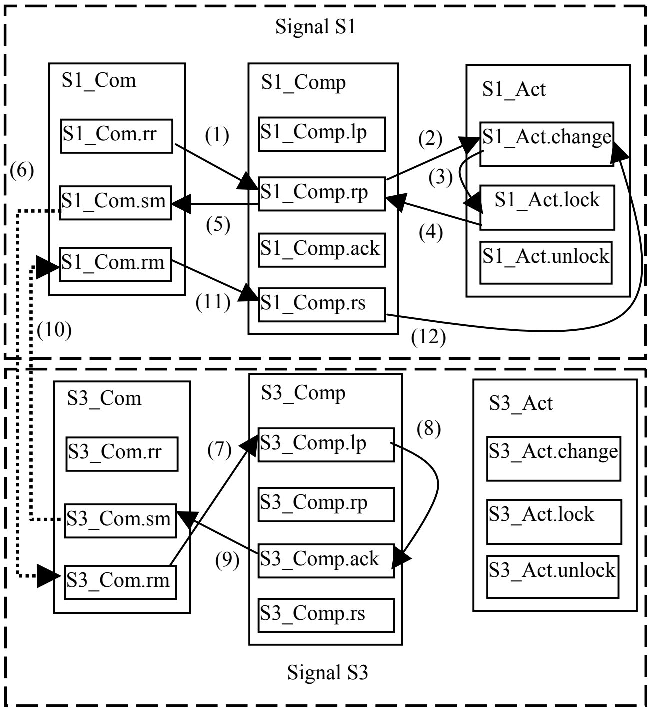

Figure 10: Invocation flow of signal S1 and S3 when X-1 is requested.

These steps are shown from step (1) to step (6). When signal S3 receives the message from network, it analyzes the message and checks whether it is the same as the current status or not. In figure 10, we assume they are the same, i.e., signal S3 is refusal status, so acknowledgment message is sent back to S1 directly. Step (7)→(10) shows this process.

When signal S1 receives the acknowledgments from all route-related devices and if they satisfy the requirements, i.e., all conditions for signal S1 changing to permission are achieved, signal S1 changes itself to permission status (display green). Step (11) and step (12) describe this process.

Step (6) and step (10) represent the communication between signal S1 and S3, which is accomplished by using a network.

Processes of other devices such as signal SS2, point P1 and P2 are similar to process of signal S3.

5 Conclusion and discussion

The proposed approach can improve greatly the reusability of function module when designing and realizing the system, and can simplify the complexity of real-time control system when analyzing these systems.

In future research, the reliability of G-nets DRIS models will be analyzed with Petri net tools and methods.

References

[1] K. Akita, T. Watanabe, H. Nakamura, I. Okumura: "Computerized Interlocking System for Railway Signaling Control; SMILE", *IEEE Trans., Ind.*, 1A-21, May 1985.

[2] Heck, B. Wills, L. and Vachtevanos G. "Software Technology for Implementing Reusable, Distributed Control. Systems", *IEEE Control Systems Magazine*, February 2003.

[3] C. Engelmann, S. L. Scott and G. A. Geist. "High Availability through Distributed Control". *Proceedings of High Availability and Performance Computing Workshop (HAPCW)*, Santa Fe, NM, USA, October 2004.

[4] M. Weiser and J. Brown, "Designing Calm Technology," *PowerGrid J.*, vol. 1, 1996.

[5] Y. Deng, S. Chang, A. Perkusich and J. de Figueiredo, "Integrating Software Engineering Methods and Petri Nets for the Specification and Analysis of Complex Information Systems". *Proc. Of The 14th International Conference on Application and Theory of Petri Nets*, Chicago, June 21-25, 1993, pages 206-223.

[6] J. de Figueiredo and A. Perkusich, "Distributed Control of Track-Vehicle System with Fault-Tolerant Characteristics: a Petri Net Based Approach", *Proc. of the IEEE international conference on Systems, Man, and Cybernetic*s, paginas,377-382,Vancouver, Canada, 1995

[7] V. Hartonas-Garmhausen, "Verification of a safety-critical railway interlocking system with real-time constraints", *Proc. of 28th Annual Int. Symposium on Fault-Tolerant Computing*, p.458, June 23-25, 1998.