

# TECHNICAL ISSUES IN MODELLING THE EUROPEAN TRAIN CONTROL SYSTEM (ETCS) USING COLOURED PETRI NETS AND THE DESIGN/CPN TOOLS

L. Jansen, M. Meyer zu Hörste, E. Schnieder

Technical University of Braunschweig,  
Institute of Control and Automation Engineering,  
Langer Kamp 8, D-38106 Braunschweig, Germany

Email: {jansen | meyer | schnieder}@ifra.ing.tu-bs.de

## Abstract

At the Institute of Control and Automation Engineering Design/CPN has been used to model the European Control System (ETCS) within a project for the Deutsche Bahn AG (German railways). This paper reports of experiences in modelling this complex, distributed automation system using Coloured Petri Nets and the Design/CPN tools. We will concentrate on some technical issues. However, for motivation we will give a brief overview of the application and will describe the modelling paradigms that we applied.

**Keywords:** European Train Control System (ETCS), Modelling, Scenarios, Distributed Simulation, Communication, Synchronization

## 1 Introduction

### 1.1 The Aims of the Interoperable European Train Control System (ETCS)

The process of European harmonization is still in progress, which also concerns the railway companies. Each European railway company has one or more train control systems, which are mostly incompatible. So every train which shall run through different countries has to be equipped with several systems or the train traction units and drivers have to be changed at every border. As a consequence, the first solution requires a complex system on-board, which leads to high installation and maintenance costs. The second one is a time consuming solution, which leads to increased operational costs. Thus, it is important to define a train control system which is standard for all countries and provides a uniform and language-independent signalling information for the man-machine-interface (MMI). [Frosig95]

Moreover, the increase of new developed trains' maximum speed demands a communication-based train control system, because at a travelling speed of more than 160 km/h the trackside

signals aren't surely recognisable any more. So for high-speed railway service the application of a communication-based train control system is necessary.

The European Train Control System (ETCS) matches these two targets: It is a communication-based train control system, which is useful for both high and low speed railway services and it defines a standard for a uniform signalling system on an MMI (Man-Machine Interface), so neither locomotives nor drivers have to be replaced when crossing borders. The textual display on the MMI can be made to comply with the driver's language.

The ETCS equipment consists of two main subsystems: Speed supervision and brake intervention are performed by the on-board system. The permission to run is given by the trackside system called "Radio Block Centre" (RBC) within a movement authority (MA). The task of this second system is to ensure that all trains in the related area are keeping the safety distance to the neighbouring trains. Also the supervision of special operational procedures like joining or splitting of trains is a task of this system. For lines with different capacities three application levels with different types of trackside and on-board equipment are possible.

## 2 Modelling the ETCS with CP-nets and Design/CPN

### 2.1 Modelling Paradigms

Modelling of control systems starts from different modelling paradigms. Before starting to model it is necessary to make sure that the methodology is suitable for achieving the modelling objective. A specific model for system functionality will be quite different from a model used for availability considerations. In modelling the European Train Control System, different modelling aspects have been integrated [JaLePtSc97]:

- components
- scenarios
- functions,

are shown on different model levels.

When modelling the component view, the focus is on communication and interaction of different subsystems. A general representation of these nets shows the subsystems and their interfaces, every subsystem being detailed on additional levels. In [JaLePtSc97] this was called the process aspect.

The scenario-based view is the modelling of operational procedures. Its main elements are the interaction between on-board and trackside equipment and the sequence of events required to maintain operation. Individual scenarios are connected to form groups and in this way they are integrated into the component model.

The functions are represented at lower model levels. The functions are specifically associated with the objects of the process aspect and represent the activities or the response to interaction requirements following from the scenarios. Some functional modules can be used in different objects and are hence modelled in so-called "functional blocks". These functional blocks are modelled as separate nets and can serve as functions in the different scenarios, without infringing

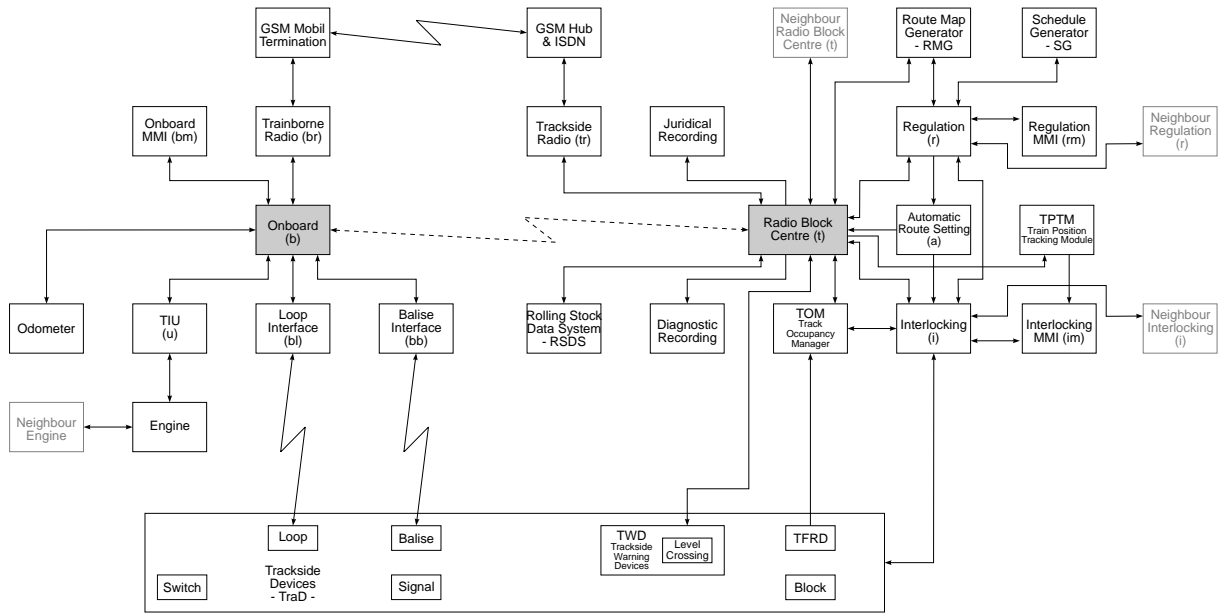


Figure 1: System Architecture of the ETCS

the modelling principle of hierarchic decomposition. This at the same time ensures that sub-nets and implementation components can be reused.

Systems modelled in this way by necessity are modular in structure – the hierarchical aspect being maintained at the same time. Because of their structure, models are easier to comprehend, easier to adapt, and they can be reused. Partitioning aspects can be considered, and symmetries can be utilised.

## 2.2 System Architecture

The goal of the first phase of our project was to model the two main components of ETCS, the onboard system and the radio block centre. Special emphasis was laid on the communication between both systems, that should be regarded according to both the more abstract interaction, represented by the sequence of telegrams, and the more functional aspects, represented by the data inside the telegrams.

Besides the onboard system and the radio block centre, there are more than a dozen of other components directly interfacing to one of the above main components or being related to them in some other way. For some of those systems, preliminary models have been built, but apart from a basic environmental model of the train none of them is fully executable by now. For an overview of the overall architecture of ETCS and the main components see figure 1. Further information on the topic can be found in [JaLeMeSc97].

For simulation purposes, we substitute the missing components by dummy solutions or by manually providing the missing stimuli directly to the interfaces of the receiving component nets.

ETCS is a modular and communication based approach to train control systems. Communication plays an important role in ETCS and therefore communication systems should be regarded as any

other kind of system with respect to structure, functionality and dynamics. By now, we modelled communication only with respect to the application layer of the well known ISO/OSI protocol stack. We did not actually model the underlying communication systems due to the restricted scope of our project, but the main component models are designed to be easily adaptable and a general framework for later supplementing of separate models of underlying communication systems to our distributed modelling is provided. This opportunity is indicated in figure 1 for the refinement of the radio link (between the onboard system and the radio block centre) by adding a radio subsystem and a GSM modul at each side.

### 2.3 Vertical Decomposition of Component Models

Structuring a component model is important for readability, maintainability and last but not least for acceptance by the user. Integrating different aspects of a system description and combining formerly isolated views on modelling is a demanding challenge for working with Petri nets, although Coloured Petri Nets and Design/CPN provide a good basis for modelling of complex systems.

For structuring the main component models of the onboard system and the radio block centre, we adopted a layered approach, proposed by [JaLePtSc97]. Dynamics and functionality are described while distinguishing between scenarios and functions. Scenarios show the behaviour of a system in it's environmental context, especially regarding the technical process and railway operations. Functions are used to process input data that has been received from external components or internal sources. In contrast to scenarios, functions do not regard processes in the environmental context and therefore can be used within arbitrary scenarios.

Functions in our sense are not restricted to the mathematical notion but can represent processes with internal states and transitions as well. To be more precise, we could call them complex functions or tasks, but we will stick to the name and instead will speak of mathematical functions or ML functions, respectively, when we have in mind the mathematical notion or their implementation.

#### 2.3.1 First Approach: Integrating Scenarios and Functions

In the beginning of the modelling we aimed at representing the application of functions within scenarios explicitly on the scenario level. Our intuition was to represent the invocation of a complex function graphically in the representation of a scenario, analogous to a function call in a functional or procedural language for sequential programming. We tried to model this aspect by making use of the hierarchy concept of Design/CPN in a straight forward way. The corresponding hierarchy or vertical decomposition, as we prefer to call it, is depicted in figure 2.

First of all, there exists a top level net  $I$  for each component model. It shows the connections of a component to other components and the corresponding (unidirectional) communication channels on an abstract level. It is possible to substitute the terminating transitions by CP-nets, that are models of the neighbouring components. But as for a distributed modelling we prefer to substitute them with interfacing functions, that realize the communication with those models implemented separately.

On the next level we have a net  $D$ , that represents the decomposition of the component model

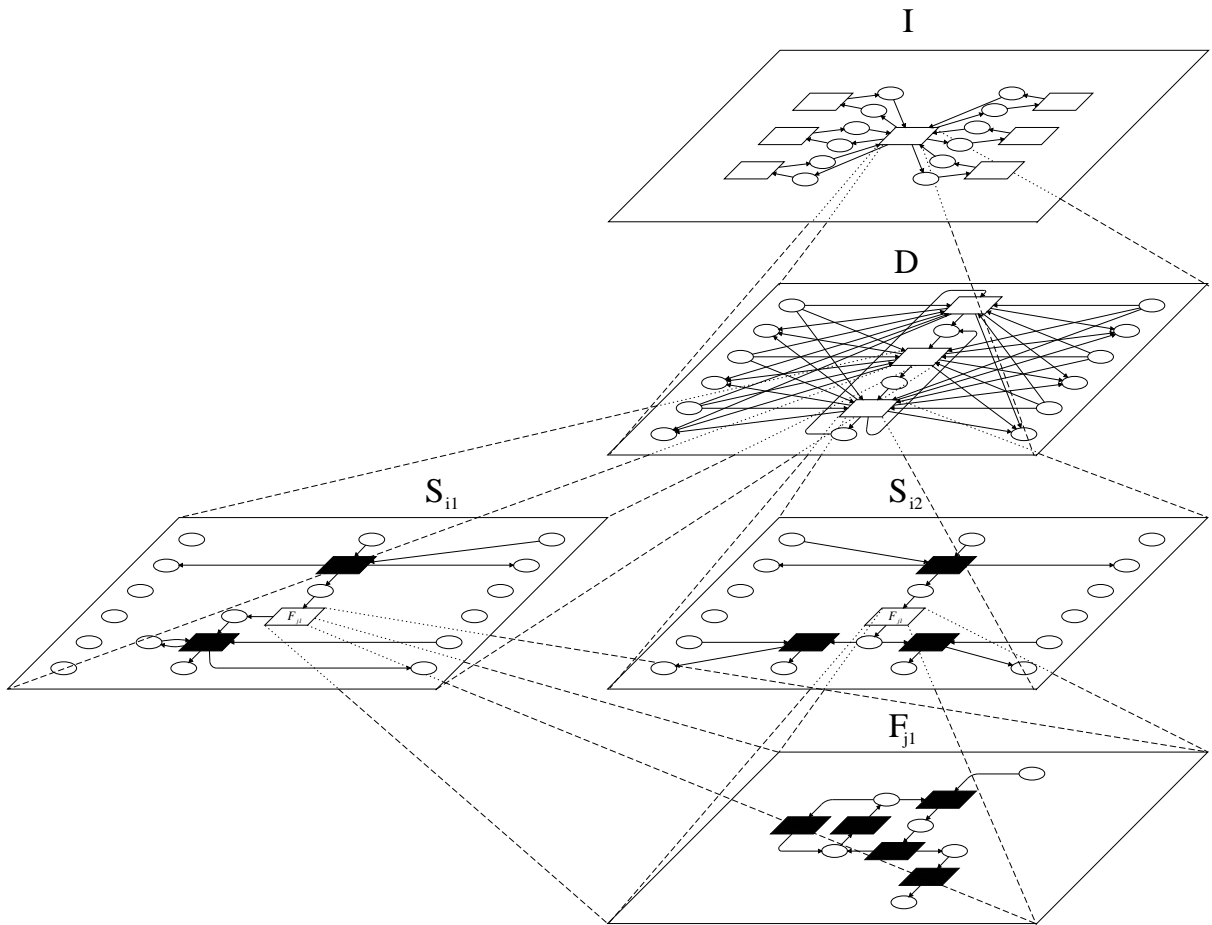


Figure 2: **Generic structure of the main component models (first approach)**. Substitution transitions are depicted white, elementary transitions black and places white. Port places are not marked up, but can be easily identified by comparing the net structures.

with respect to relevant scenarios. This decomposition may be reached traversing several sub-levels, that are combining scenarios to so called *scenario groups*.

The two bottom levels are used for representing scenarios or functions, respectively. A net, representing a scenario  $S_i$  is called a *scenario net*. A net representing a function  $F_j$  is called a *functional block*. Both may be further decomposed. A scenario net  $S_i$  that makes use of a function  $F_j$  contains a substitution transition, establishing the connection to the corresponding functional block.

Later on, this approach has come out to be against our intuition, as the semantics of hierarchical refinement within Design/CPN is of an instantiating kind. That is, if in the Design/CPN editor the same subpage, e. g. the top level net of a functional block  $F_j$ , is connected to several superpages (each with one or more corresponding supernodes), e. g. the scenario nets  $S_{i1}$  and  $S_{i2}$ , the effect after switching to the Design/CPN simulator is that the subpage is instantiated with the number of supernodes, resulting in a 1:1-relation between the supernodes and instances of the corresponding subpage. For further details concerning the concept of hierarchy in Design/CPN

see [Jensen92].

At a first quick glance, we identified the problem as a need of sharing a local process by a number of other processes, sometimes even at the same time. The relation between the shared process and those latter processes is of an orthogonal kind and, thus, unusual for hierarchical decomposition. Within Design/CPN the problem can be solved in two ways:

- declaring all local places as fusion places in the shared net with the effect of synchronizing all its instances;
- not sharing the common net but implementing the calling and returning of the functional block via communication over a special fusion place.

The decision was made for the second solution, mainly because of the extensive use of fusion places required for the first solution, and second because of tool problems with Design/CPN 2.0, encountered at that time, with managing a restricted number of fusion places.

We think that with respect to the point of integrating (complex) functions and scenarios in a Petri net model, a semantics of shared subtrees would better accomplish our needs in this special case, although it probably would compromise the strict hierarchical decomposition paradigm. However, analyzing the abstract nature of this “irregularity” in terms of hierarchical decomposition is an open matter, important for integrating different views and paradigms for modelling with hierarchical Petri nets in general.





### 2.3.2 Improved and Extended Approach: Coordinating Szenarios and Functions

The improved generic structure of the main component models is depicted in figure 3. According to the afore mentioned problem of integrating scenarios and functions, and aiming at the preferred solution, we adopted the generic structure. Now functions are not any more subordinated to scenarios but, speaking in terms of structure, coordinated to scenarios. The dynamic coordination is managed by means of a newly introduced fusion place. This place serves as an internal message channel for the calling of complex functions and returning results to scenario nets.

Comparing with the first approach, the model was restructured mainly effecting the decomposition level formerly denoted as  $D$ . Besides the decomposition of scenarios  $D_S$  the decomposition of functions  $D_F$  is now represented explicitly. The net  $D$  is used to distinguish between these two decomposition structures.

In the course of this restructuring we have introduced a relaxation of another modelling constraint, that a function net should not be directly connected to the component’s interface. The schematic representation of the function  $F_j$  slightly differs from that one in figure 2 in adding the interface places to the function net. This shall indicate, that a function may directly receive/send data to/from the component’s interface.

For pragmatcal reasons we admit a paradigmatic shift concerning the relation between scenarios and functions. Functions  $P_k$  for preprocessing incoming messages and functions  $P'_k$  for postprocessing outgoing messages that both are closely related to a certain interface  $k$  are represented separate from the other application functions  $F_j$ . For this purpose an additional layer was introduced between the top level and the decomposition level. The corresponding net  $A$  now represents all the functionality and dynamics of a component.

Legend:			
	place / port place	$T_0$ general test net	$I$ interface net (top level)
	fusion place	$T_k$ interface related test net	$A$ application net
	substitution transition	$P_k$ preprocessing net	$D$ decomposition net
	elementary transition	$P'_k$ postprocessing net	$S$ scenario net
			$F$ function net

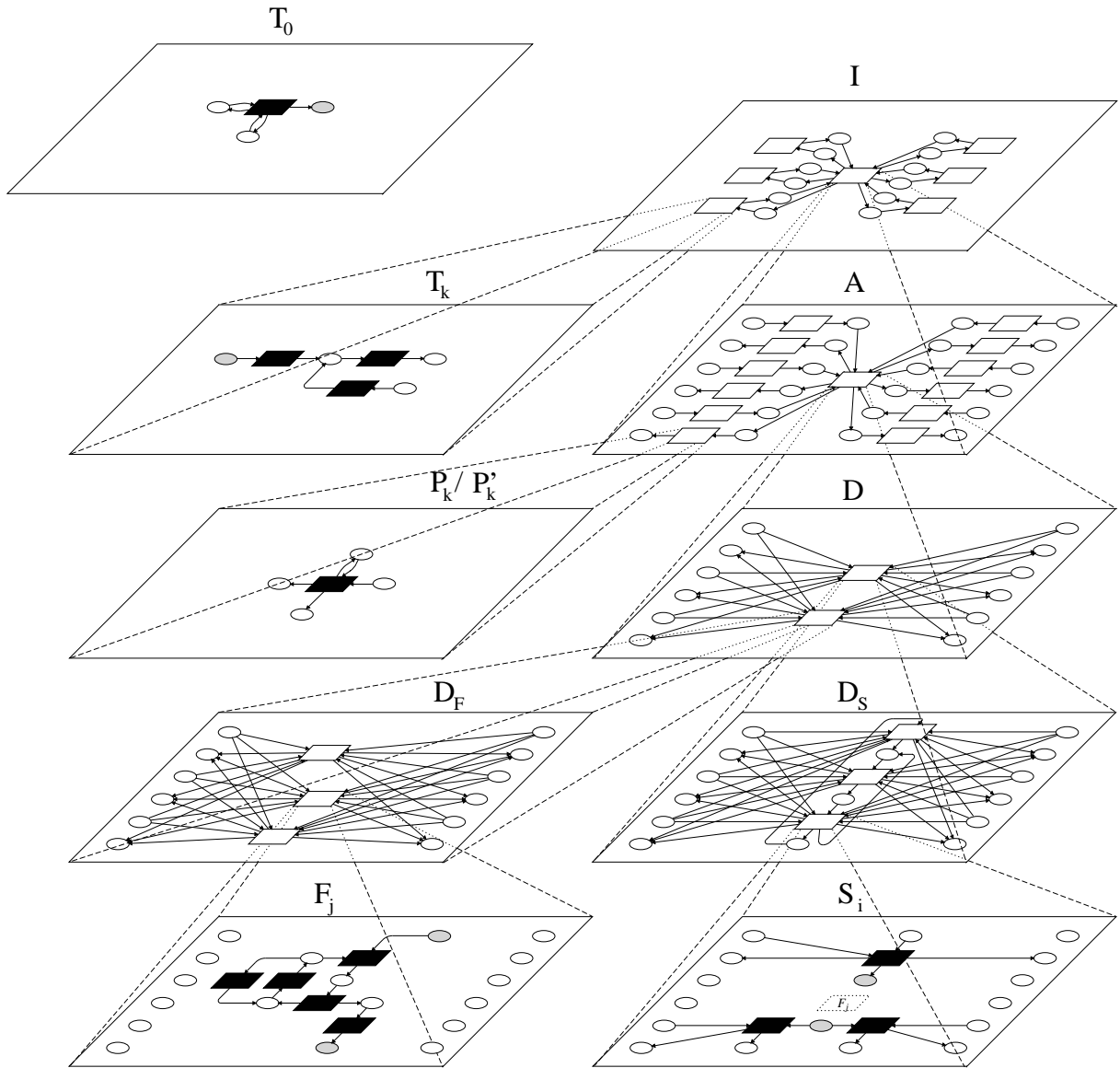


Figure 3: **Generic structure of the main component models (improved and extended approach)**. Substitution transitions are depicted white, elementary transitions black, fusion places grey and other places white. Port places are not marked, but can be easily identified by comparing the net structures.

Moreover, the nets  $T_0$  and  $T_k$  have been added for preliminary simulation purposes. The latter nets  $T_k$  substitute the terminal transitions of the top level net, i. e. one for each neighbouring component or its corresponding interfacing system, respectively. They implement special testing interfaces for producing stimuli, i. e. incoming messages, with respect to corresponding interfaces of certain neighbouring components. The net  $T_0$  is used to coordinate the nets  $T_k$  with regard to a sequence of stimuli, that can be manually defined in  $T_0$ . The stimuli are executed step by step while incrementing the simulation clock after each stimulus has been processed.

## 3 Technical Issues

### 3.1 Distributed Modelling

In terms of nets, a *distributed model* can be viewed as a Petri net whose net graph is not connected. There are pairs of nets with no path from one net to the other net and vice versa, their sets of places are disjunct and they have no fusion places in common. We call these nets *component nets*. The easiest way in order to obtain a distributed model is to cut the *whole net* at places, where the modelled system can be decomposed into components. Components are weakly connected (i. e. by communication) and do not share memory or control flow. For the cutting purpose, those places should be selected which represent communication channels at the border (interface) of a component to its surrounding. (Remark: Throughout this paper we only regard channels for unidirectional point-to-point communication). The selected places first are duplicated. Then it is decided, which place shall belong to which component net. The places should be renamed according to their assigned component while the arcs from or to transitions of the opposite component net have to be deleted.

### 3.2 Distributed Simulation

For a distributed simulation, we execute the separate component process models by means of separate simulation tasks. The simulation tasks themselves are programs that are run on operating systems and underlying hardware. So, first of all, we have to distinguish between the two words: process and task. We use the words in the way that we call the execution of a CP-net within the Design/CPN simulator a (simulation) task whereas the sequence of occurrences of transitions in a CP-net model is called a process.

The above component CP-nets can be separated by assigning each component net to a Design/CPN diagram. Thus, loading these Design/CPN diagrams to different Design/CPN tasks and starting them to simulate, we get a *distributed model* running on a *distributed system*, e. g. a cluster of workstations. As we formerly noted, the whole net was cut at the component's interfaces. Thereby the communication channels in the whole model were broken up and, in addition, the component models were assigned to different simulation tasks. As a result, we have to supply a communication mechanism on the level of Design/CPN simulation tasks, enabling two Petri nets to exchange data while being executed. We call this *implemented communication*. It must not be confused with the *modelled communication* being subject to the modelling of the application.

As a second aspect, distributed systems are generally characterised by the absence of a *global time*. Time is measured by clocks belonging locally to the tasks. There may be several kinds of

clocks concerning hardware, operating system and simulation tasks, each measuring a different concept of time. Tasks are distributed over different hardware, they run on different operating systems, they show different work load and share their resources (processor, memory, communication) with other tasks. Thus, we have to deal with the effect that distributed simulation tasks generally will not run synchronously without additional measures. This raises two questions: In which cases is synchronization of simulation tasks necessary? And if so, how can it be done? These questions shall be answered in the section about synchronization mechanisms.

### 3.3 Communication between Seperate CP-Nets

In order to have separate CP-net simulation tasks communicating with each other in a distributed simulation run, a family of basic communication protocols was defined.

The protocols offer three service primitives:

**SETUP:** for establishing a logical communication channel, generating the corresponding transfer file and setting up the connection to the other component model.

**SEND:** for sending a telegram to another system.

**TERMINATE** for closing down the connection.

A common protocol instance for all defined protocols is about to be realized for Design/CPN. Its implementation consists of seven hierarchical CP-nets and some ML routines. The protocol instance may be reused by defining a supernode for each interfacing component. Failures occurring during the execution of service primitives are detected and an error indication is returned to the service caller.

The protocol instance has been implemented and successfully tested for a first simple protocol, enabling a static point to point connection without the need of setting up or terminating the connection.

Generally, the communication is implemented via reading and writing to a common file system, accessible for all simulation tasks. Logical communication channels are mapped to physical files in the file system. These files are called *transfer files*. Some further files are used for synchronizing the protocol instances for more advanced protocols which are not implemented yet.

As a means of communication files have been preferred for two main reasons: First, the communication via files can be easily observed externally and can be specifically manipulated for simulating communication errors, and second, nearly every simulation tool offers a run-time interface to the file system. Thus, transferring of data via files opens the opportunity to implement the protocol for other tools so as to enable a combined simulation in a diverse tool environment.

### 3.4 Synchronization

Regarding a distributed simulation with time, the aim of synchronization is to establish a common control over the distributed flow of simulated time. The simulated time is measured by clocks, one for each simulation task. These clocks now have to be addressed as *local clocks*, one for each component simulation being run on a Design/CPN simulator or any other simulation

tool. The synchronization mechanism depends on the representation of time. There are three alternatives:

1. no time
2. discrete time (normally with equidistant time intervals)
3. event time (with non-equidistant time intervals)

The first alternative leads to a causal model, not considering timing aspects at all. There is no need of a timed simulation neither for modelling nor for implementing the communication between Petri nets. Under these circumstances synchronization is no matter of concern.

The second alternative leads to a time based model. The behaviour of a system is computed for predetermined times and thus can be observed according to a schematic time grid. For a distributed simulation a common time base is defined the local clocks have to be synchronized with. For defining the time base a trade off between accuracy and efficiency is inevitable since both dimensions have their optima in the opposite extremes, maximum accuracy for a small time base and maximum efficiency for a huge time base.

The third alternative is mentioned for completeness but was not yet studied in our project and therefore will not be considered in this paper.

### 3.5 Synchronization Mechanism for Discrete Time

For the distributed model execution we developed a preliminary version of a synchronization mechanism for discrete time simulation, adopting the method of *barrier synchronization* (see [Lange97]) for CP-nets and Design/CPN.

The mechanism is based on a central process serving the global time to the local Design/CPN clocks. These clocks are maintained by the distributed Design/CPN simulation tasks. The synchronization of a Design/CPN simulation is realized by adding a special net page to each component net. This page contains a so called client process, that realizes the synchronization for the component net. The clients half of the synchronizing process is identical for each component net and apart from its synchronizing aspect does not interfere with the rest of the component net.

The mechanism consist of four steps performed cyclically as follows:

1. The server process sends a synchronization signal via a special synch message channel to the other simulation tasks.
2. The client process within each component simulation task polls to this synch message channel, until it receives the synchronization signal.
3. When there are no more transitions enabled for a client process under the current value of the local clock, its value is incremented towards the next discrete time. Among the newly enabled transitions there is at least one that makes the client process send back a synchronization acknowledgement to the server process and restarts the polling to the synch message channel for the next synchronization signal.

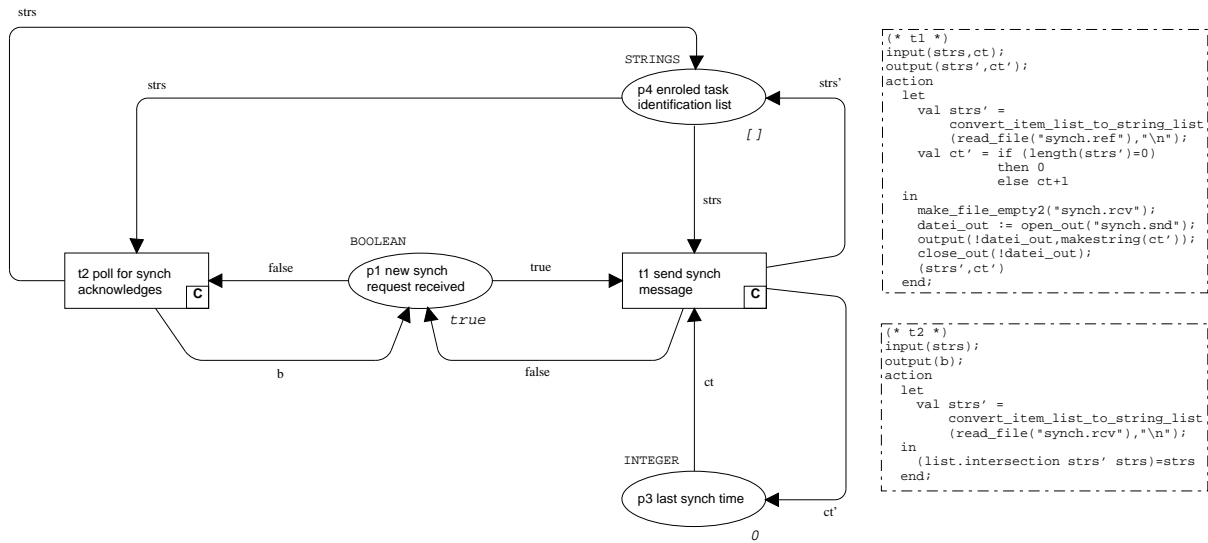


Figure 4: CP-net with server process for synchronization

4. The server process waits until the acknowledges of all client processes have been received and then sends the next synchronization signal. Thus the cycle begins from the start again.

An implementation of the server process for this synchronization mechanism with CP-nets is depicted in figure 4. The implementation uses three files: *synch.snd* as the synch message channel, *synch.rcv* for the synch acknowledgements of the client processes and *synch.ref* containing the list of identities of all enroled simulation tasks as a reference for comparison. The aim of the file *synch.ref* is to cope with dynamically adding and completing of simulation tasks.

The CP-net consists of two transitions and three places. Transitions *t1* and *t2* implement the first and fourth of the afore mentioned steps. In the initial state, the place *p3* contains an initial token with default value 0 for the synchronization time. There are no enroled simulation tasks known, as represented by an initially empty list in place *p4*, and the place *p1* contains a token with the value *true*, which means that a new synchronization signal has to be sent to all the enroled client processes.

The sending of the synchronization signal is done by transition *t1*. It updates *p4* with the list of enroled simulation tasks read from the file *synch.ref*, deletes old synch acknowledges from the file *synch.rcv* and writes the new synchronization message to the file *synch.snd*. The message contains the actual synchronization time that is incremented from the last stored value in place *p3* if there are any simulation tasks enroled at all. The transmission of the synchronization time with the synch signal is intended for later use regarding event time. Finally a token with the value *false* is generated in the place *p1* which disables transition *t1* and enables transition *t2*.

The transition *t2* cyclically reads (i. e. polls) the file *synch.rcv* for acknowledges from the client processes. An acknowledge consists of the identification of the corresponding simulation task. As each client appends it's task identification to the file *synch.rcv*, successful completion of the current synchronization request can be checked by comparing that file to the list of enroled task identifications in place *p4*. Transition *t2* stores the result of this comparison in place *p1*, and in

case of successful completion the next synchronization cycle can begin.

It should be obvious, without giving the exact declaration of the used colorsets, that the implementation of the server process does not require timed simulation. However, for the client process, a timed simulation is necessary. As the net structure is quite similar to that of the server process, we will not give a graphical net representation of the client process. The list of identities of the enrolled simulation tasks is not used for the client process. The transition, which is sending the synch acknowledge, has to generate a delayed token with an incremented time stamp.

## 4 Conclusion and Outlook

For the application of formal analysis methods to the ETCS, two models have been developed. The partition of the model follows the partition of the real system, both the on-board and the trackside system are represented each by one hierarchical model. Like the real system these models have interfaces where the data-exchange can be observed. The nets are divided into different types according to their position in the structure and their content. The component models are implemented separately, so different CP-nets may be simulated on separate workstations. A simulation runs through a sequence of scenarios which call several functions.

In the actual state the two models consist of circa 200 nets and 2500 transitions and places. The application logic is supplemented by some constructs which perform supporting tasks like time synchronization, stimuli generation or implementing communication between separate CP-nets. Also a smaller model has been developed, which is simulating the environment, i.e. the track, the physical train etc. Two more models of the interlocking and the regulation are still in working progress.

**Future Plans** The aim of our formal model is

1. to check the completeness of the specification of the ETCS,
2. to use it for a systematic derivation of test cases and
3. to evaluate at an early stage the specification of the european stardized interfaces of ETCS according to the national railway environment.

The future work will be directed to identifying ways of using the models for hardware in the loop tests with prototypes of the main components. Another aspect of intent is the derivation or automatic generation of code for an implementation of the modelled components.

## References

- [Frosig95] P. Frosig. ETCS: Requirements to Seamless Cross-Border Operation. *International Railway Journal and Rapid Transit Review*, 9 (September), 1995, New York.

- [JaLeMeSc97] A. Janhsen, K. Lemmer, M. Meyer zu Hörste, E. Schnieder. Migration Strategy for Different Level of the European Train Control System to Existing Railway Environment. In *Proc. of WCCR 97 – World Congress of Railway Research, Volume C: Power Supply, Signalling, Telecommunications and Non-conventional Systems*, pages 335–341, Florence, 1997.
- [JaLePtSc97] A. Janhsen, K. Lemmer, B. Ptok, E. Schnieder. Formal Specification of the European Train Control System. In M. Papageorgiou and A. Pouliezios, editors, *Proc. 8th IFAC Symposium on Transportation Systems*, pages 1215–1220, Chania, 1997. IFAC.
- [Jensen92] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1. Springer-Verlag, 1992.
- [Lange97] K.-J. Lange. On the Distributed Realization of Parallel Algorithms. In F. Plasil, K. G. Jeffery, editors, *Proc. of the 24th Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM'97), Milovy, Czech Republic, November 1997*, pages 37–52. Springer, Berlin, LNCS 1338, 1997.