Temporal Analysis of Data Flow Control Systems

C. Bernardeschi[a] A. Bondavalli[b] Gy. Csertán[c] I. Majzik[c] L. Simoncini[a]

[a] *Department of Information Engineering, University of Pisa, Pisa, Italy*

[b] *CNUCE-CNR, Pisa, Italy*

[c] *Department of Measurement and Instrument Engineering, Technical University of Budapest, Hungary*

Abstract

Due to their distributed/parallel and data-driven nature, control systems can easily be modeled according to a data flow approach. Control systems are very often real-time systems therefore a formalism able to capture timing is required. In this paper we introduce a data flow model that includes time and priority for specifying real-time control systems and we give its formal semantics. The control system is specified by a data flow network which, beside the controller, may include the model of the plant at some abstraction level. Time is associated to any computational activity and time accounting is made directly in the model and not as a separate issue. Priorities allow to deal with events, as alarm signals, which cannot be delayed. A general framework for the indirect evaluation of the model is introduced, and a data flow network to timed Petri net transformation is defined allowing the utilization of the automatic tools of Petri nets for analyzing the temporal properties of the data flow network. The approach is illustrated by an example in which, after the application of the transformation, selected performance measures are computed.

Keywords: Control systems; control system design; control system analysis; time-domain analysis; data flow model; timed Petri nets.

1 Introduction

A control system is made up of a plant with sensors and actuators, a controller and the interface towards an operator. The controller continuously interacts with the plant through control signals and with the operator. The controller

[⋆] A preliminary version of this paper appeared in the 12th IFAC Workshop on Distributed Computer Control Systems, DCCS-94 executes the control algorithm processing the parameters of the environment sent as signals by the sensors and sends signals to the actuators to intervene in the environment. In simple systems the software realizing the functions of the controller is usually organized as a cycle of instructions executed sequentially. However, this approach does not scale well to more complex systems where (i) such an ordered cycle may take too long for the real-time requirements to be satisfied, (ii) many of the controller instructions are independent and (iii) the asynchronous nature of the system implies that very few activities described in the cycle actually take place at each round. Thus asynchronous models, being able to describe as high degree of parallelism as is admitted by the requirements of the application, should be used to specify and design such systems.

Among others, the data flow approach has been considered to model asynchronous control systems [19]. Following the data flow paradigm, a system design is represented by a network of nodes which execute concurrently and exchange data by asynchronous message-passing. Each node is associated with a set of possible activities. A given activity of a node starts upon receipt of a pre-defined set of messages, executes a computation and terminates with the output of messages toward other nodes. Although the control engineer's specification may be a direct representation of an *abstract* analog computer, or of hard-wired logic, i.e. of time-continuous functions, it is easily mapped into a data flow net with nodes that accept input messages and produce output messages, to be executed cyclically at some appropriately high repetition rate [5]. Data flow models have the advantages of a simple graphical representation (data flow networks), compactness and expressiveness of the parallelism inherent in the modeled system [4,27].

Early timing analysis plays an important role in the development process of control systems, contributing at the validation to be performed as early as possible in the design process itself. In case of real-time systems, where response time of the system is constrained by the specification, the temporal analysis of the system is essential for determining the satisfaction of the requirements. The maximum execution and/or response time must be provided. A temporal analysis is nevertheless very important also for systems that are not required to satisfy real-time requirements. A designer, especially in the early stage of the development, would like to know which is the expected time performance of the design, being prepared to accept also rather rough measures. The average response time, the average execution time and the steady state analysis are of interest.

To model control systems, we have extended the traditional data flow paradigm, showing how it can be used in the early phases of the system design for specification and verification purposes. To analyze the design, we propose an indirect technique, by transforming the data flow model into an equivalent representation which can be analyzed directly. At this extent, a transformation is introduced from data flow networks toward timed Petri nets. The transformation is proved to generate a Petri net which permits the indirect timing analysis of the data flow model. The idea of using Petri nets in the field of performance evaluation of data flow networks was first applied in [20], in which only a subset of data flow networks has been considered, composed of a few types of different data flow nodes.

The rest of the paper is organized as follows. In Section 2, we justify the data flow approach and compare it with other approaches, then formally define our model. In section 3 the example we use throughout the paper is introduced and described using our formalism. In Section 4 we briefly describe a class of timed Petri nets and the transformation from the data flow model to this class of Petri nets. The relation of the data flow model and the corresponding Petri net is discussed. In Section 5 the transformation is applied to the example and an analysis of the average and worst case temporal behavior and the schedulability of the simple control system introduced in Section 3 is performed. Section 6 discusses the limits of this kind of modeling and evaluations.

2 Data flow design of control systems

Following the data flow design approach, the controller is modeled by a data flow network that interacts with the external environment. Sometimes, the external environment, both the plant and the operator, may be modeled together with the controller to obtain a closed network. Generally, the plant is specified at a very high abstraction level, by considering the load of the controlled system and generating the input/output signals accordingly.

We focus on *event triggered* control systems [22], systems in which activities start as a reaction to asynchronous events as they occur in the external environment. An alternative is *time triggered* systems [21], represented by systems which periodically observe the state of the external environment. Event triggered control systems are tailored for a wider set of load hypotheses and offer better efficiency, although not allowing an easy validation as time triggered ones. Moreover, in a broad class of control systems, the computation is driven by the change of the state of signals, which are often on/off signals. This allows us to restrict to *uninterpreted* data flow nets: data items are not distinguished from each other, they are represented by tokens. A transition of a signal is represented by an item in the corresponding channel of the data flow net.

2.1 Data-flow design environment

Efforts undertaken to support the design of control systems aim at providing integrated environments in which the composition of the model, simulation based examination of its behavior as well as formal analysis of the properties are possible, moreover, the production of prototypes is supported by automatic synthesis tools. These environments are either based on a single modeling formalism with a consistent semantics or seek to systematically combine disjoint semantics which fit to different modules or phases of the design.

In the literature, various system description techniques and formalisms are known, with different goals and application areas. Among others, we can mention Hatley-Pirbhai [13] and Ward-Mellor [30] diagrams as general techniques for structured development of real-time systems or SDL [6] as a standard description language for communication systems. To model asynchronous, event triggered systems, mostly the wide family of Petri nets is proposed. In particular, the application of plain Petri nets has the main advantage that a widely accepted semantics and sophisticated tools are available. However, it results in very large and complex models, not always well dominated by the designer. Using hierarchical nets [16], a more compact model can be provided, but the analysis techniques are usually based on the mapping of the hierarchical model to a plain one. Coloured Petri nets [15] have the advantage that, beside providing a compact model, data dependent control flow and computation can be modeled. Safety properties can be analyzed using sophisticated tools, but the tools for timing analysis are now in the early phases of development.

The data flow paradigm is also received some attention and it is generally considered as an appropriate basis of integrated design environments. The most noticeable tools are Ptolemy [7] (used for hardware-software co-design and the design of control systems), some commercial frameworks for signal processing applications and tool sets based on synchronous data flow languages (Signal [11], Lustre [12]). Data flow approach has been also considered as appropriate means of modeling asynchronous control systems [19] and real-time processing [25,29]. Data flow models proved to be especially useful in designing embedded control systems where hardware as well as software elements (decomposed in further steps of model refinement) have to be modeled and analyzed together. Namely, data flow nodes can be considered either as high-level abstractions of software modules containing several tasks with well-defined interfaces to the other ones, or as hardware modules describing activities which can be implemented e.g. by finite state machines.
Although research on data flow is less visible and obtained not such amount of results as that e.g. on the family of Petri nets, this paradigm still has potentials that justify its investigation.

In particular, a data flow network is usually very close to the intuitive representation of the specification of a system, as conceived by a control engineer. It is easy to understand and fits to the designers' way of thinking, therefore time to construct and understand the model can be reduced (the support of the re-use of nodes and subnetworks also contributes in it). As the automatic analysis and synthesis tools become more and more powerful and the design time is dominated by the model construction phase, it can result in significant speed-up in the design process. The hierarchical representation and the expressive power of nodes result in a more compact model which highlights the structure and helps the designer to focus on given subsystems and activities.

Beside the intuitive description, the possibility to assign well-defined semantics enables the formal verification of data flow modeled systems. Many proposals of data flow formalisms are known, which fit to different application areas [17,18]. However, a disadvantage of the data flow paradigm originates from this variety, as the lack of a single, widely accepted model results in the shortage of evaluation tools. However, the lack of automatical tools can be resolved by applying the same approach as in the case of other high level formalisms: the model can be evaluated indirectly, providing transformations towards underlying simpler formalisms that can be analyzed by existing, sophisticated packages.

Following this approach, the data flow model can be used efficiently as an expressive, high level common representation of the design environment. The indirect analysis resolves the lack of direct tools and also enables a more extensive analysis of the model since it allows to combine the results related to different aspects which could not be obtained by a single (simpler) formalism. Since the transformation of the model and the back annotation of the results into the original environment is automatical, the designer does not have to deal with the particular formalisms and interfaces used in the underlying packages. Integration of new tools can be solved by providing the necessary model transformation and remapping, without having the designers to learn the new tool itself.

To analyze different properties of the design, different underlying formalisms and tools can be selected. As two examples, let us consider safety and timing analysis (dependability analysis techniques were also extended to be used in data flow modeled systems [10]). Safety analysis of the model can be solved by transformation of the data flow model either to a process algebra specification or to Petri nets. Using the process algebra formalism, equivalence relations and logic checking tools are provided to check the satisfaction of requirements [2]. Petri nets are also suitable to analyze safety when the set of states reachable by the execution of the system can be obtained. To derive the timing properties of the model, the Petri net based analysis fits best [9], since a wide range of sophisticated tools are available.

According to the above considerations, our design environment is structured as follows. A graphical data flow editor allows the designer to develop the model in a quick and convenient way by supporting hierarchical composition of nodes and sub-networks, utilization of application-specific libraries and also automatic tools to get predefined connection schemes (e.g. redundant structures) [8]. The basis of model analysis, i.e. the common representation of the design environment is a plain network of data flow nodes, which is assigned a well-defined semantics allowing to define various, theoretically well-grounded transformations toward other representations. The designer does not have to deal with this plain net, as it is provided by the data flow editor resolving the subnetworks and library elements automatically. According to the analysis method requested by the designer, the common representation is transformed to the formalism of the underlying tool that can perform the analysis. The tool is then invoked and the results are propagated back into the environment in the terms of the original design.

In the current paper we focus on a sub-problem of the design environment, the timing analysis of the design. After introducing the plain data flow net (used as the common internal representation of the environment) and its formal semantics, we define the transformation toward timed transition Petri nets and prove that it preserves the timing properties. This way we are allowed to use it for the indirect timing analysis of the model.

2.2 Timed data flow networks

To model dataless, event-triggered, asynchronous control systems we had to extend the traditional formalism of data flow nodes. To study the state-dependent behavior of the modeled system, we have introduced *states* of the nodes. Each node can execute various activities, referred to as *firings*. Accordingly, a particular state is the *working state*: a node is in the working state during the time in which it is executing a firing. Firings are executed sequentially, one at a time. The selection of the firing to be executed depends on the state of the node and on the availability of tokens over the input channels of the node. To manage situations when activities are enabled together, but some priority constraint exists between them, *priorities* are assigned to the firings of the nodes. Additionally, to be able to investigate the timing properties, the firings are associated with a *timing parameter*. The timing parameter denotes a delay for the execution of the activity. When the plant is modeled in the design, time may be associated also with the activities of the plant.
Our approach, since within the nodes of the network there is a finite state machine like representation of states and state transitions, allows to express data driven parallel computing as well as state-dependent response of the system.

Definition 1 (node) *A node n is a tuple $n = (I_n, O_n, S_n, R_n)$ where:*

> I_n *- set of input channels*
> O_n *- set of output channels*
> S_n *- set of states; sw_n is the* working state *of the node*
> R_n *- set of firings where $r \in R_n$ is a tuple $r = (s, X_{in}, s', X_{out}, \tau, \pi)$*
>> $s, s' \in S_n \setminus \{sw_n\}$ *- states before and after the firing, respectively*
>> $X_{in}(c) \mapsto I\!N$*, for each $c \in I_n$ - number of tokens removed from each input channel*
>> $X_{out}(c) \mapsto I\!N$*, for each $c \in O_n$ - number of tokens put on each output channel*
>> τ *gives the delay for the execution of the firing*
>> π *is the priority of the firing.*

A firing $r = (s, X_{in}, s', X_{out}, \tau, \pi)$ is *enabled* if and only if the node is in the state s and each input channel $c \in I_n$ contains at least $X_{in}(c)$ tokens. If a firing is enabled then it may be executed.
A firing is *firable* if it is at the highest priority level among the enabled ones. If there are more enabled firings at the highest priority level then the firable one is selected randomly. If a node is in a non-working state and none of its firing is enabled then the node is *idle*.

A firing $r = (s, X_{in}, s', X_{out}, \tau, \pi)$ of a node n is executed during the interval of time delimited by two instantaneous events:

- a *Start-event* (denoted by $s(r)$) in which $\forall c \in I_n, X_{in}(c)$ tokens are removed from the input channel c of the node, and the state of the node changes from s to the working state. Moreover a delay is sampled according to τ.
- an *End-event* (denoted by $e(r)$) which occurs when the sampled delay expires. With this event, $\forall c \in O_n, X_{out}(c)$ tokens are deposited into the output channel c of the node, and the state of the node changes from the working state sw_n to s'.

The input channels of the node n in Figure 1(a) are a and b ($I_n = \{a, b\}$), while c is the output channel of n ($O_p = \{c\}$). If a firing r exists with initial state s, final state s', time of the execution of the firing τ', priority π' and such that it removes i tokens from a, j tokens from b and outputs k tokens on the channel c ($X_{in}(a) = i, X_{in}(b) = j$ and $X_{out}(c) = k$) the notation for the firing is the following: $r = (s, [a \rightarrow i, b \rightarrow j], s', [c \rightarrow k], \tau', \pi')$. If the number of tokens put or removed by the firing is zero, then it is not included in the notation.

Definition 2 (network) *A data flow network DFN consists of a set of nodes, on condition that each channel occurs at most once as input and at most once as output channel of a node.*
Given a network, it is denoted by a tuple $DFN = (N, C, R, \sigma^0)$, where N de-

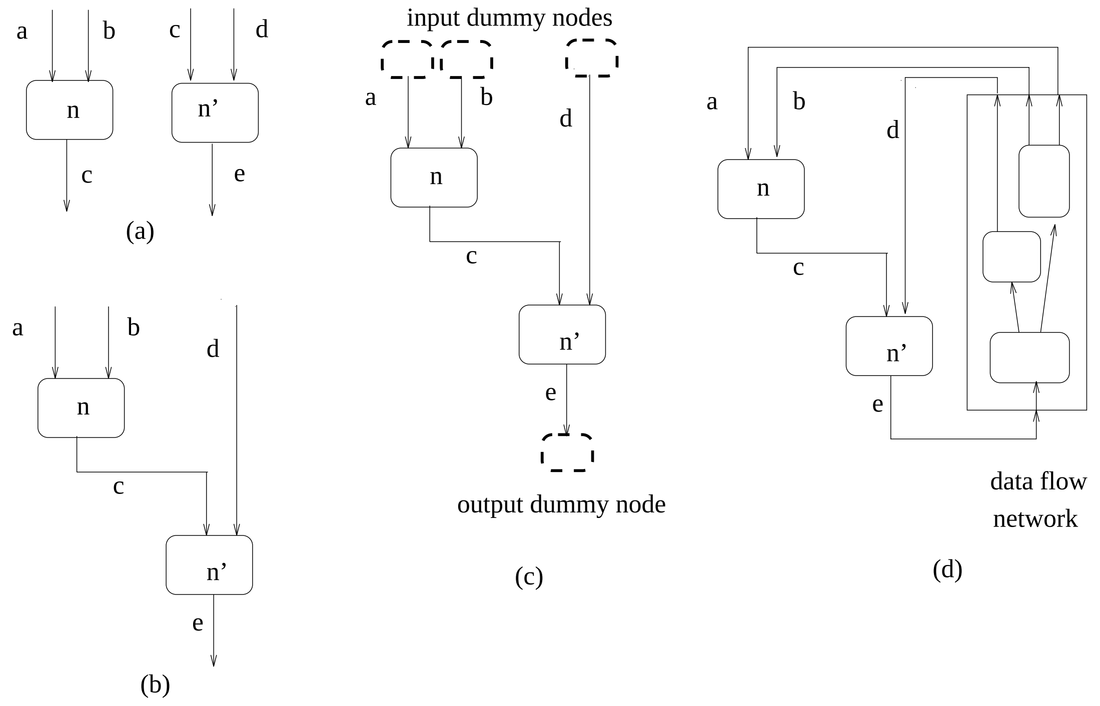

Fig. 1. (a)The node n and n'. (b) The network composed by n and n'. (c) Environment modeled by dummy nodes. (d) Environment modeled by a net.

notes the set of nodes, C denotes the set of channels while the set of firings is referred to as R. Let Σ be the set of states of the network. A state $\sigma \in \Sigma$ consists of the states of the nodes ($\sigma(n)$ for each node n) and the states of the channels ($\sigma(c)$ for each channel c, denoting the number of tokens in the channel). The initial state of the network is denoted by σ^0.

We can obtain *open* or *closed* networks. A network is *open* if some channels are not on both ends connected to the nodes. For example channels a, b, d and e in Figure 1 (b). In the case of *closed* networks, each channel links two nodes.

The environment can be modeled in two different ways:

(i) The input and output of tokens is simulated by adding *dummy nodes* to the network, graphically represented by dotted boxes as depicted in Figure 1(c). The activity of an input dummy node is to deposit tokens on an input channel of the network, while an output node removes tokens from an output channel.
(ii) The input and output of tokens is modeled with another data flow network representing the environment. The composition of the two networks results in a closed net, as shown in Figure 1(d).

We apply the following policy in the network for the execution of firings:

- If there are firable firings in the network then one of them is selected randomly. Note that firable firings belong to different nodes. Then the Start-event of the selected firing is executed and the corresponding node enters the working state by sampling a delay. This step is repeated until firable firings exist, the system time is not increased.
- If there are no firable firings then the End-event of the firing which has the minimum remaining delay θ is executed. If two or more firings sampled the same minimum delay then the selection is random, the firing that is not chosen will have a zero remaining delay in the next iteration. The system time is increased with θ.

To avoid to record each small state change, we are interested in selecting a subset of the states encountered during a computation such that it is still representative of the computation itself and that allows to maintain all the relevant information for studying the timing behavior of the system. Consider that the execution of End-events represents the output of tokens towards nodes and the system time may be increased only during the execution of an End-event, while the execution of a Start-event never increases the system time. Therefore we abstract from states in which Start-events are executed and characterize computations by recording only those states in which an End-event is executed. For doing this we introduce the concept of vanishing and tangible state of the network, similar to vanishing and tangible states of Petri nets [24]. A state is called *vanishing* if there is at least one firable firing in the net, otherwise it is called *tangible*.

Note that after execution of the End-event of a firing the state of the net may be tangible or vanishing. Let consider the case of a vanishing state. By definition, the firable firings belong to different nodes. Since firings of different nodes are independent from each other (by definition of the network), the execution of the Start-events of these firings in any order results in the same tangible state of the net.
Since in a node the selection among enabled firings being at the same highest priority level is random, different sets of firable firings can be taken into account in a vanishing state. To eliminate this inner nondeterminism, each firing of a node should have an unique priority.

We define the computation of the net as a series of tangible states. Since the initial state of the network may be vanishing or tangible, we have to distinguish two cases.

Definition 3 (computation) *A computation of the network is a finite or infinite sequence* $\sigma^0 \overset{(e^0,F_e^0,\theta^0)}{\rightsquigarrow} \sigma^1 \overset{(e^1,F_e^1,\theta^1)}{\rightsquigarrow} \cdots \sigma^k \overset{(e^k,F_e^k,\theta^k)}{\rightsquigarrow} \sigma^{k+1} \cdots$, *where*

- σ^0 *is the initial state;*
- $\forall k \geq 1, \sigma^k$ *is a tangible state of the net;*

– *for each tangible state* σ^k,
  · e^k *is the End-event of a firing;*
  · F_e^k *is the set of Start-events leading to a new tangible state of the net after the execution of* e^k*;*
  · $\theta^k \in I\!R \cup \{0\}$*, is a time delay.*
– *if* σ^0 *is vanishing then* $\sigma^0 \overset{(e^0,F_e^0,\theta^0)}{\rightsquigarrow} \sigma^1$ *is replaced by* $\sigma^0 \overset{(F_e^0,\theta^0)}{\rightsquigarrow} \sigma^1$ *where*
  · F_e^0 *is the set of Start-events of firings which are firable in* σ^0
  · $\theta^0 = 0$*;*

The one-step computation $\sigma^k \overset{(e^k,F_e^k,\theta^k)}{\rightsquigarrow} \sigma^{k+1}$ is interpreted as follows: θ^k delay after the previous End-event or after the zero system time for $k = 0$, the state of the net changes from σ^k to σ^{k+1} by the execution of the End-event e^k and by the execution of the Start-events of F_e^k in any order. If σ^0 is vanishing, the first tangible state is reached by executing the Start-events of the firings which are firable in the initial state (the system time in not increased).

A tangible state σ of the net is *reachable* if and only if a computation exists that transforms the initial state σ^0 into σ.

3 The train set example

In this section we consider the train set example described in [28], where trains move unidirectionally along a circuit divided into sections (Figure 2). With the assumption that the train's length is less than each section's length, a safety criterion for the movement of trains requires that there must be at least one free section between the head of any two trains in order to avoid collision.

A reservation system can be used to this purpose: a train reserves always two sections for itself. One section is occupied by the head of the train and a second one is reserved behind the first. Moreover, to be allowed to move forward, a train has to reserve the next section, so, for limited time intervals, it has three sections reserved.

The system is divided into two subparts, the plant and the controller. We modeled the plant as a data flow network, thus obtaining a closed network (Figure 3 for 6 sections). Section $SECT_i$ is responsible for sending sensor signals to the controller and for receiving actuator signals from the controller. When a train enters a section the sensor sends an `es` signal to inform the controller. After receiving the `ls` signal by the controller the actuator lets the train proceed to the next section. At the same time a signal `sn` is sent to the next section to model the movement of the train. The first part of the controller, nodes CNT_i, where CNT_i is associated to $SECT_i$, releases section $i \ominus 2$ (signal `rel`) and tries to reserve section $i \oplus 1$ (signal `res`) ($\oplus$ and $\ominus$ denote

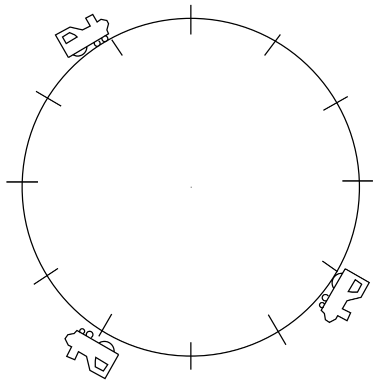

Fig. 2. The train set example

the modulo–n addition and subtraction, respectively, where n is the number of the sections). If the reservation is successful then CNT_i sends the `ls` signal to the section. The second part of the controller, nodes RES_i, keeps track the reserved and free sections. Receiving a `rel` signal it releases the section, receiving a `res` signal it reserves the section by sending the `ok` signal to the CNT_i node. Of course if a given section is reserved for a train it can not be reserved for another one. The timing variables of the example are reported in Table 1.

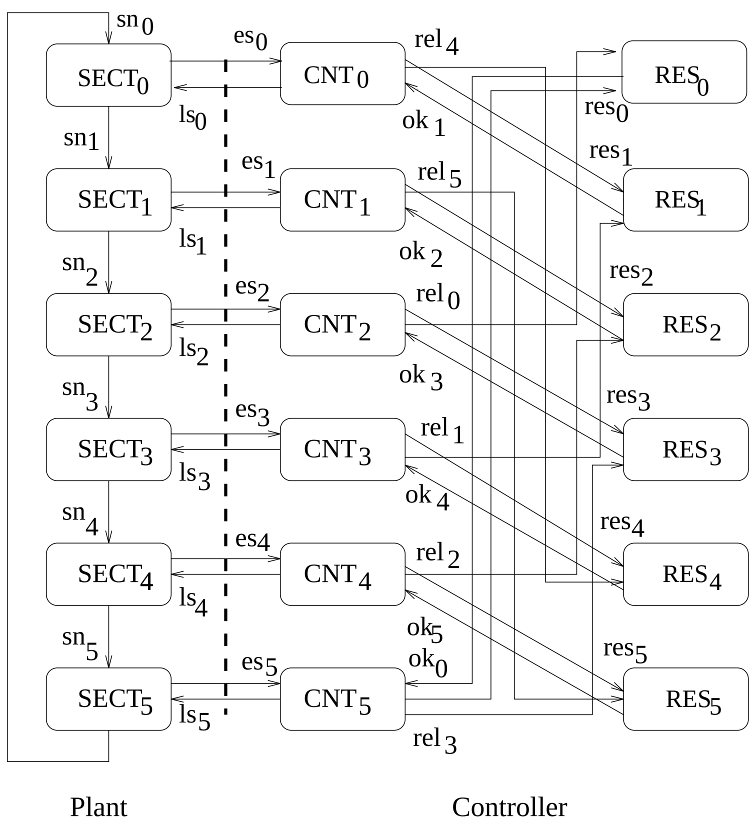

Fig. 3. Data flow model of the train set example

Table 1
Timing variables of the example

timing variable
τ_{sen} - time consumed by a sensor sending a signal
τ_{cross} - time a train needs to move along the section
τ_{act} - time spent by receiving an actuator signal
τ_{cnt} - time the controller needs to send signals
τ_{res} - time for reserving a section
τ_{rel} - time for releasing a section

The resulting data flow specification, when the circuit is divided into six sections and two trains initially located into section 0 and section 2, respectively, are running over the circular track is the following.

$N = \bigcup_{i=0}^{5}\{SECT_i, CNT_i, RES_i\}$

where

$SECT_i:$
$I_{SECT_i} = \{sn_i, ls_i\} \qquad O_{SECT_i} = \{sn_{i\oplus 1}, es_i\}$
$S_{SECT_i} = \{s_i, s'_i, s''_i\}$
$R_{SECT_i} = \{f_i = (s_i, [sn_i \to 1], s'_i, [es_i \to 1], \tau_{sen}, 0),$
$f'_i = (s'_i, [], s''_i, [], \tau_{cross}, 0),$
$f''_i = (s''_i, [ls_i \to 1], s_i, [sn_{i\oplus 1} \to 1], \tau_{act}, 0)\}$
$CNT_i:$
$I_{CNT_i} = \{es_i, ok_{i\oplus 1}\} \qquad O_{CNT_i} = \{ls_i, res_{i\oplus 1}, rel_{i\ominus 2}\}$
$S_{CNT_i} = \{u_i, u'_i\}$
$R_{CNT_i} = \{g_i = (u_i, [es_i \to 1], u'_i, [res_{i\oplus 1} \to 1, rel_{i\ominus 2} \to 1], \tau_{cnt}, 0),$
$g'_i = (u'_i, [ok_{i\oplus 1} \to 1], u_i, [ls_i \to 1], \tau_{cnt}, 0)\}$
$RES_i:$
$I_{RES_i} = \{res_i, rel_i\} \qquad O_{RES_i} = \{ok_i\}$
$S_{RES_i} = \{v_i, v'_i\}$
$R_{RES_i} = \{r_i = (v_i, [res_i \to 1], v'_i, [ok_i \to 1], \tau_{res}, 0),$
$r'_i = (v'_i, [rel_i \to 1], v_i, [], \tau_{rel}, 0)\}$

Initial state of the channels:
$\forall c \in C \setminus \{res_1, res_3\}, \sigma^0(c) = 0;\ \sigma^0(res_1) = \sigma^0(res_3) = 1$

Initial state of the nodes:
$\sigma^0(SECT_i)_{i=1,3,4,5} = s_i;\ \sigma^0(SECT_i)_{i=0,2} = s'_i$
$\sigma^0(CNT_i)_{i=1,3,4,5} = u_i;\ \sigma^0(CNT_i)_{i=0,2} = u'_i$
$\sigma^0(RES_i)_{i=0,1,2,5} = v'_i;\ \sigma^0(RES_i)_{i=3,4} = v_i$

In [2] it has been shown that the previous data flow design guarantees a safe behavior of the trains along the track. The proof has been done for the data flow description of the controller without timing parameters, using process algebras and model checking verification technique.

4 Analyzing timed data flow networks

To analyze temporal properties of data flow networks, we use Timed Transition Petri nets (TTPN), which are Petri nets in which random execution delays are associated with transitions [3]. This formalism has been accepted and widely used for performance evaluation, resulting in the availability of a set of automatic tools.

We define the transformation from data flow networks to TTPN so that no restrictions are put on the distribution of the firing delay of our DFN. In order to take advantage of the availability of automatic tools here we focus on the *Deterministic and Stochastic Petri Nets* (DSPN,[24]), a subclass of TTPN. DSPN permits the association of deterministic and exponentially distributed execution delays of transitions. The exponential distribution, due to its memoryless property, leads to an isomorphism between DSPN and continuous-time Markov chains, which highly simplifies the model analysis. Thus, DFNs restricted to either deterministic or exponentially distributed firing delays may be analytically evaluated, while DFNs including other distributions must be analyzed resorting to simulation. It is worth mentioning that there is active research on analytical models and methods for coping with non-exponential firing delays [14]. Hopefully this will result in the production of new tools able to deal with non-exponential distributions. These tools can be directly used by applying our transformation and will allow to evaluate more general networks analytically.

In the following, we first recall the usual definition of DSPN introducing some new definitions used in our environment (details on the semantics and firing policy can be found in in [24]), then we define our transformation and examine the relation of the two nets.

4.1 Background

Definition 4 *A Deterministic and Stochastic Petri Net is a tuple* $DSPN = (P, T, D, W, H, M^0, \Pi, ,)$, *where:*

P *- set of places*

T - *set of* immediate *or* timed *transitions*
$D \subseteq P \times T \cup T \times P$ - *set of directed arcs, called flow relation*
$W : D \rightarrow \mathbb{N}^+$ - *weight function of the arcs*
$H \subset P \times T$ - *set of inhibitor arcs*
$M^0 : P \rightarrow \mathbb{N}$ - *initial marking*
$\Pi : T \rightarrow \mathbb{N}$ - *priority function*
$, \; : T \rightarrow \{\mathbb{R}^+ \cup \{0\}\}$ - *time function in the case of timed transitions;*

For timed transitions, , is the deterministic time value or the parameter of the negative exponential distribution. For immediate transitions , is used for the selection of the transition to fire (random switch, if more enabled transitions are at the same highest priority level).

The current distribution of tokens over the places denotes the marking of the net. Formally, a marking is a mapping $M : P \rightarrow \mathbb{N}$ which gives the number of tokens $M(p)$ for each place p in the network. Tangible and vanishing markings were defined similarly like in the DFN.

For $t \in T$, ${}^\bullet t = \{p \mid pDt\}$ is called the *preset* (also *input places*) of t, $t^\bullet = \{p \mid tDp\}$ is called the *postset* (also *output places*) of t.
A transition t is *enabled* if $\forall p \in {}^\bullet t,\ M(p) \geq W(p,t)$. Two transitions t and t' are in conflict if ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$.
A timed transition t is *firable* if it is continuously enabled during its whole execution time, sampled according to $,(t)$. An immediate transition is *firable* if it is at the highest priority level among the transitions being enabled.

The firing policy of the net is *race with enabling memory* [24]. In a vanishing marking, enabled transitions being at the highest priority level are found and one of them is selected on the basis of the random switches. It is fired and this way a new marking is reached, the system time is not increased. When a new tangible marking is entered, each enabled timed transition samples a delay. The sampled distributions are the remaining time to fire, counting the time for which the transition was enabled since it has last become enabled. The minimum sampled delay θ determines both the transition t that will be executed and the sojourn time in the marking. The system time is increased by this minimum delay θ and t is fired, thus reaching a new marking. If multiple transitions sampled the same minimum delay then one of them is selected randomly. The others will sample zero delay in the next tangible marking.

In defining a computation of the DSPN, we abstract from vanishing markings concentrating on the sequence of tangible markings.

Definition 5 (computation) *The* computation *of the DSPN is defined as follows:*

$M^0 \overset{(t^0,F_t^0,\theta^0)}{\leadsto} M^1 \cdots M^k \overset{(t^k,F_t^k,\theta^k)}{\leadsto} M^{k+1} \cdots$, *where*

- *M^0 is the initial marking;*
- *$\forall k \geq 1, M^k$ is a tangible marking of the net;*
- *for each tangible marking M^k,*
  - *t^k is a timed transition;*
  - *F_t^k is the set of immediate transitions fired according to the firing policy after t^k, resulting in a new tangible marking*
  - *$\theta^k \in \mathbb{R} \cup \{0\}$ is a time delay.*
- *if M^0 is vanishing then $M^0 \overset{(t^0,F_t^0,\theta^0)}{\leadsto} M^1$ is replaced by $M^0 \overset{(F_t^0,\theta^0)}{\leadsto} M^1$ where*
  - *F_t^0 is a set of immediate transitions fired subsequently before reaching the tangible state M^1*
  - *$\theta^0 = 0$;*

The one-step computation $M^k \overset{(t^k,F_t^k,\theta^k)}{\leadsto} M^{k+1}$ is interpreted as follows: θ^k delay after the firing of the previous timed transition or after the zero system time for $k = 0$, the marking of the DSPN changes from M^k to M^{k+1} by the execution of the t^k timed transition and then by the firing of the immediate transitions in F_t^k.

4.2 The DFN to DSPN transformation

The transformation $\mathcal{T} : (N, C, R, \sigma^0) \to (P, T, D, W, H, M^0, \Pi, ,)$ maps a DFN to a DSPN by mapping the events of the data flow net onto transitions of the Petri net and both the channels of the net and the states of the nodes onto places of the Petri net.

(i) **Places**
Let $P = P_C \cup P_S \cup P_W$, where P_C are the places corresponding to channels, P_S the places corresponding to normal states and P_W the places corresponding to working states obtained by the following rules:
- each channel is mapped to a different single place with the name equal to the name of the channel
  $\forall x \in C : \mathcal{T}(x) = x \in P_C$;
- for each node, any normal state is mapped to a different single place with the same name of the state;
  $\forall x \in \bigcup_{n \in N}(S_n \setminus \{sw_n\}) : \mathcal{T}(x) = x \in P_S$;
- for each node n, the working state is mapped onto a set of places; more precisely a place is associated with each firing of the node and the name of the place is $sw_n^r, r \in R_n$;
  $\forall n \in N : \mathcal{T}(sw_n) = \{sw_n^r, r \in R_n\}$. Thus $\bigcup_n \mathcal{T}(sw_n) = P_W$;

(ii) **Transitions, priority Π and time parameter ,**

- the Start-event of each firing r is mapped to an immediate transition with the same name of the event and priority inherited from the firing. Since in this case , denotes a random switch, it is set to 1 for all immediate transitions to have equal probability for the execution of transitions with the same priority:
  $\forall r \in R : \mathcal{T}(s(r)) = s(r) \in T$ with $s(r)$ an immediate transition;
  $\Pi(s(r)) = \pi$, where π is the priority associated to r; , $(s(r)) = 1$;
- the End-event of each firing r is mapped to a timed transition with the same name of the event and zero priority and the time function inherited from the firing:
  $\forall r \in R : \mathcal{T}(e(r)) = e(r) \in T$ with $e(r)$ a timed transition; $\Pi(e(r)) = 0$; , $(e(r)) = \tau$.

(iii) **Flow relation**

- Transitions representing Start-events of firings are connected to places representing states and channels (input places) and working states (output places):
  For each $t \in T$ if $t = \mathcal{T}(s(r))$ with $r = (s, X_{in}, s', X_{out}, \tau, \pi)$, $r \in R_n$:
  $^\bullet t = \{p \in P_C \mid p \in I_n \text{ and } X_{in}(p) \neq 0\} \cup \{s\}$ and $t^\bullet = \{sw_n^r\}$;
  $\forall p \in {}^\bullet t \cap P_C : W(p,t) = X_{in}(p)$, while $W(s,t) = 1$ and $W(t, sw_n^r) = 1$.
- Transitions representing End-events are connected to places representing working states (input places), normal states and channels (output places):
  For each $t \in T$ if $t = \mathcal{T}(e(r))$ with $r = (s, X_{in}, s', X_{out}, \tau, \pi)$:
  $^\bullet t = \{sw_n^r\}$ and $t^\bullet = \{p \in P_C \mid p \in O_n \text{ and } X_{out}(p) \neq 0\} \cup \{s'\}$;
  $\forall p \in t^\bullet \cap P_C : W(t,p) = X_{out}(p)$, while $W(sw_n^r, t) = 1$ and $W(t, s') = 1$.

(iv) **Initial marking**

- Places corresponding to channels are set according to the initial state of the channel in the DFN:
  $\forall c \in P_C$: $M^0(c) = \sigma^0(c)$;
- The marking of places corresponding to states is always 0, except for the places corresponding to the initial state of the nodes which is equal to 1:
  $\forall x \in P_W$: $M^0(x) = 0$;
  $\forall x \in S_n, n \in N$: if $\sigma^0(n) = x$ then $M^0(x) = 1$ else $M^0(x) = 0$

An example of the transformation is reported in Figure 4, where the Petri net corresponding to the data flow node CNT_1 is shown. Places are represented by circles with the name of the place inscribed inside; moreover, the places in $P_W \cup P_S$ are represented by dotted circles. Transitions are represented by boxes; moreover, immediate transitions are represented by shadowed boxes. The name of the transition is inscribed inside the box. The weight associated to an arc is represented graphically as a number located near the arc (we omit the number when it is equal to 1).

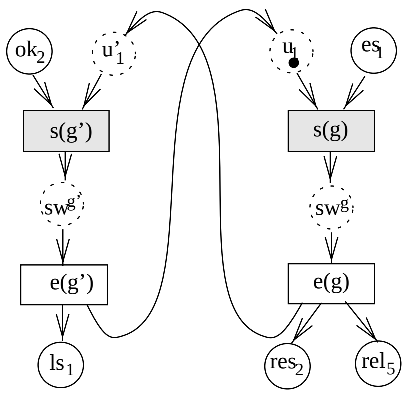

Fig. 4. Representation of the node CNT_1

By the transformation, there is a one to one mapping between immediate transitions of the DSPN and Start-events of the DFN. Similarly there is a one to one mapping between timed transitions of the DSPN and End-events of the DFN.

A marking of the DSPN is mapped to a state of the DFN by the following definition:

Definition 6 (State corresponding to a marking) *Given a marking M reachable from the initial marking of the DSPN. The state σ of the DFN corresponding to M is obtained as follows:*

$\forall c \in C, \sigma(c) = M(c)$
$\forall n \in N, \sigma(n) = s \in S_n \setminus \{sw_n\}$ *if* $M(s) = 1$*;*
$\qquad \sigma(n) = sw_n$ *if* $\exists r \in R_n$ *such that* $M(sw_n^r) = 1$

The mapping is unambiguous, since exactly one of the places of a node in $P_S \cup P_W$ contains a token (by the construction of the net).

A state of the DFN corresponds to a set of markings of the DSPN which differ in the distribution of tokens in the places of P_W. The reason is that there is a single working state of a node in the DFN, but there is a place corresponding to the working state of a node for each firing of the node in the DSPN.

4.3 Properties of the transformation

The defined transformation provides the DSPN in its initial marking corresponding to the initial state of the DFN. In this section we prove that reachability and timing analysis problems of the DFN can be solved using the DSPN model. It has to be shown that each computation of the DSPN can be uniquely mapped to an existing computation of the DFN, and then that each computation of the DFN corresponds to an existing computation of the DSPN. Here only the sketch of the proof is outlined, the detailed description is found in [26].

Theorem 7 *The DFN model is isomorphic with the DSPN model in the following sense:*
1) a tangible marking M^n of the DSPN is reachable by a computation
$$M^0 \overset{(t^0,F_t^0,\theta^0)}{\rightsquigarrow} M^1 \cdots M^{n-1} \overset{(t^{n-1},F_t^{n-1},\theta^{n-1})}{\rightsquigarrow} M^n$$
if the tangible state σ^n corresponding to M^n is reachable in the DFN by a computation
$$\sigma^0 \overset{(e^0,F_e^0,\theta^0)}{\rightsquigarrow} \sigma^1 \cdots \sigma^{n-1} \overset{(e^{n-1},F_e^{n-1},\theta^{n-1})}{\rightsquigarrow} \sigma^n,$$
with $t^i = \mathcal{T}(e^i)$, $F_t^i = \mathcal{T}(F_e^i)$ and σ^i is corresponding to M^i for all i in the computation, the θ^i parameters are equal as denoted. (To simplify the notation, we define $\mathcal{T}(\{s(r_1), \cdots, s(r_n)\}) = \{\mathcal{T}(s(r_1)), \cdots, \mathcal{T}(s(r_n))\}$.)

2) Conversely, a tangible state σ^n of the DFN is reachable by a computation
$$\sigma^0 \overset{(e^0,F_e^0,\theta^0)}{\rightsquigarrow} \sigma^1 \cdots \sigma^{n-1} \overset{(e^{n-1},F_e^{n-1},\theta^{n-1})}{\rightsquigarrow} \sigma^n$$
if there is a marking M^n reachable in the DSPN by a computation
$$M^0 \overset{(t^0,F_t^0,\theta^0)}{\rightsquigarrow} M^1 \cdots M^{n-1} \overset{(t^{n-1},F_t^{n-1},\theta^{n-1})}{\rightsquigarrow} M^n,$$
with $t^i = \mathcal{T}(e^i)$, $F_t^i = \mathcal{T}(F_e^i)$ and σ^i is corresponding to M^i for all i in the computation; θ^i parameters are equal as denoted.

Proof sketch. *The proof of the Theorem is inductive on the length of the computation.*
(a) For the initial marking of the DSPN and for the initial state of the DFN, the Theorem is valid (by definition of the transformation and the mapping). They are reachable by zero-length computations.
(b) Let assume that the Theorem is valid for marking M^{n-1} of the DSPN reached from M^0 by the computation
$$M^0 \rightsquigarrow \cdots \overset{(t^{n-2},F_t^{n-2},\theta^{n-2})}{\rightsquigarrow} M^{n-1}$$
and for state σ^{n-1} of the DFN reached from σ^0 by the computation
$$\sigma^0 \rightsquigarrow \cdots \overset{(e^{n-2},F_e^{n-2},\theta^{n-2})}{\rightsquigarrow} \sigma^{n-1}$$
Then, the following statements can be derived:

(i) *A timed transition t is enabled in M^{n-1} if and only if the firing, whose End-event is e with $\mathcal{T}(e) = t$, is actually being executed by the DFN in σ^{n-1}.*

(ii) *A timed transition t with a firing delay θ can be selected to be fired in M^{n-1} if and only if the End-event e, $\mathcal{T}(e) = t$, with the same delay θ can be selected to be executed in σ^{n-1}.*

(iii) *Let M' be the marking reached from M^{n-1} by firing an enabled timed transition t and let e be the End-event such that $\mathcal{T}(e) = t$. Then, the state σ' reached from σ^{n-1} by executing e is the state corresponding to M'.*

In the following, we distinguish two cases:

(a) If M' is tangible, then σ' is tangible as well, in this way a new tangible marking and a new tangible state are reached, the induction proves the Theorem as $M^{n-1} \xrightarrow{t} M'$ with $M^n = M'$, $\sigma^{n-1} \xrightarrow{e} \sigma'$, with $\sigma^n = \sigma'$.

(b) If M' is vanishing, then σ' is vanishing, too. In M', the firing of firable transitions leads to a new tangible marking. Similarly, in σ' the execution of firable Start-events leads to a new tangible state.
The sets of firable transitions that can lead to a new tangible state are exactly the sets of transitions which can be derived by transforming the sets of Start-events leading to a new tangible state in the DFN. Additionally, the probability that transitions of a given set are fired (each after the other, according to the firing policy of the DSPN) is equal to the probability that in the DFN the Start-events of the corresponding set are executed. If the transitions of a given set are fired then the reached tangible marking M'' is such that: when executing the corresponding set of Start-events leads to σ'' in the DFN, it is exactly the state corresponding to M''. The induction proves the Theorem as $M^n = M''$, $\sigma^n = \sigma''$.

Applying the above statements, point 1) of the Theorem can be proved for each possible computation of the DSPN. A similar reasoning can be applied to prove point 2). □

In the proof of Theorem 7 the type of the distribution of the firing delays distributions is not utilized; the Theorem is valid for a DFN and a corresponding Petri net with generally distributed firing times, using the semantics and firing policy defined in Section 2.2 and in Section 4.1. Additionally, the stochastic behavior of the two models are equivalent in the sense that in a computation of a DSPN, for each tangible marking, the probability that the model evolves to a given successor tangible marking is the same as the probability that in the corresponding computation the DFN evolves to the corresponding next tangible state.

Theorem 7 and its consequences assure that reachability analysis (focussed on tangible states) as well as steady-state and transient timing analysis of the DFN can be performed using the corresponding DSPN model. One can investigate e.g. existence (or absence) of given states, how much time is needed to reach a given tangible state, what are the computations leading to a given state, what is the probability of a state or execution rate of a firing (if a steady state exists).

Finally we note that in the Petri net given by the transformation, the timed transitions are not in conflict with each other, so the policy of the net for resolving conflicting timed transitions is irrelevant. Tools and simulators which use race with age memory firing policy are also suitable for the analysis of temporal properties of the data flow network.

5 Timing analysis of the train set example

The train set example is used to highlight the usefulness both of the data flow design approach and of the indirect analysis. Instead of performing the complete analysis of the train set controller (which is not the main concern of this work), we report a few examples to show the kind of analysis that can be done on the design. Among various timing characteristics interesting for a designer, and choosing the default values of the timing parameters (exponential distribution) as reported in Table 2, we show the following evaluations:

- The average cycle time of trains: this is a performance indicator and will be computed using analytical solutions for the Petri net.
- The maximum cycle time of trains: this quantity needs to be computed in order to verify if timing requirements are satisfied; it will be derived by simulation (using stochastic variables with exponential distribution implies that each transition can experiment an infinite worst case time).
- The probability that a train has to wait at the end of a section for the signal of the controller: this is a reactive parameter that will be computed again using analytical solutions for the Petri net.
- Schedulability analysis: in the DFN model the nodes execute parallel, but further steps of the model refinement require to deal with the supporting hardware architecture. The activities of the nodes have to be associated with processes which are allocated and scheduled on processors. It is important to analyze as early as possible the expected timing behavior of the system when the parallel execution is constrained.

Table 2
Parameters and their values

parameter:	value: [1/s]
λ_{cross}	0.01
λ_{sen}	50
λ_{act}	$\lambda_{sen}/3$
λ_{cnt}	$\lambda_{sen} * 10$
λ_{res}	$\lambda_{sen} * 10$
λ_{rel}	$\lambda_{sen} * 10$

Simulations were performed using the SimNet Petri net simulator, while SPNP [23] (accuracy is better than 10^{-10}) has been used for the analytical solutions. We summarize the cost of the analysis at the end of the section.

5.1 The Petri Net Equivalent to the Data Flow Net

The Petri net derived by applying the transformation to the train set data flow design is shown in Figure 5. To keep the Figure as simple as possible, each couple of immediate transition and timed transition (associated to each firing of a data flow node by the transformation) is replaced by a single timed transition.

In our design environment, the Petri net equivalent of the model is generated automatically [1]. The DFN is composed using a graphical data flow editor [8], where the nodes of identical type are used as modules, although a simple textual editor could be used directly. Note that also the representation using our low level data flow formalism (18 nodes) is simpler than the corresponding Petri net which consists of 84 transitions (42 immediate and 42 timed ones) and 120 places. (Generally, if the number of section is s then the number of places in the Petri net is $20s$, the number of transitions is $14s$.) It would be more difficult to manually derive and maintain the Petri net model even in this very simple example. The data flow model is more compact and easy to survey, expresses the (natural) structure of the problem.

5.2 Average Execution Time

To compute the average time a train needs to cover the whole circuit steady-state analysis was performed. The results of steady-state analysis provide the average number of tokens in places and the average throughput of transitions. From these values the cycle time is computed in the following way: the average throughput of firing $f0$ (which denotes the sending of a sensor signal when a train has entered section 0) gives the number of sensor messages sent in unit time from section $SECT_0$ to the controller, that is the frequency trains enter $SECT_0$. Since (i) the number of trains is fixed and (ii) the safety rule imposes that trains can not overtake each other, in one cycle $f0$ will fire once for each train. Therefore the cycle time is: $\tau_{cycle} = n/throughput(f0)$ where n is the number of trains. Since the plant is a ring the same result can be obtained fixing the observation point at the entrance of any section.

The first analysis performed aims at verifying that, when the time necessary to trains to cross sections (τ_{cross}) is several orders of magnitude larger than the time necessary for the controller, the cycle time remains almost constant and close to the time trains need to cross a circuit of the same length without the controller. λ_{sen} has been selected to range between 1000 and 0.01 (1/seconds) that covers not only plausible values but also unrealistic ones. This choice allows to measure the impact of the delay of the controller and to find for

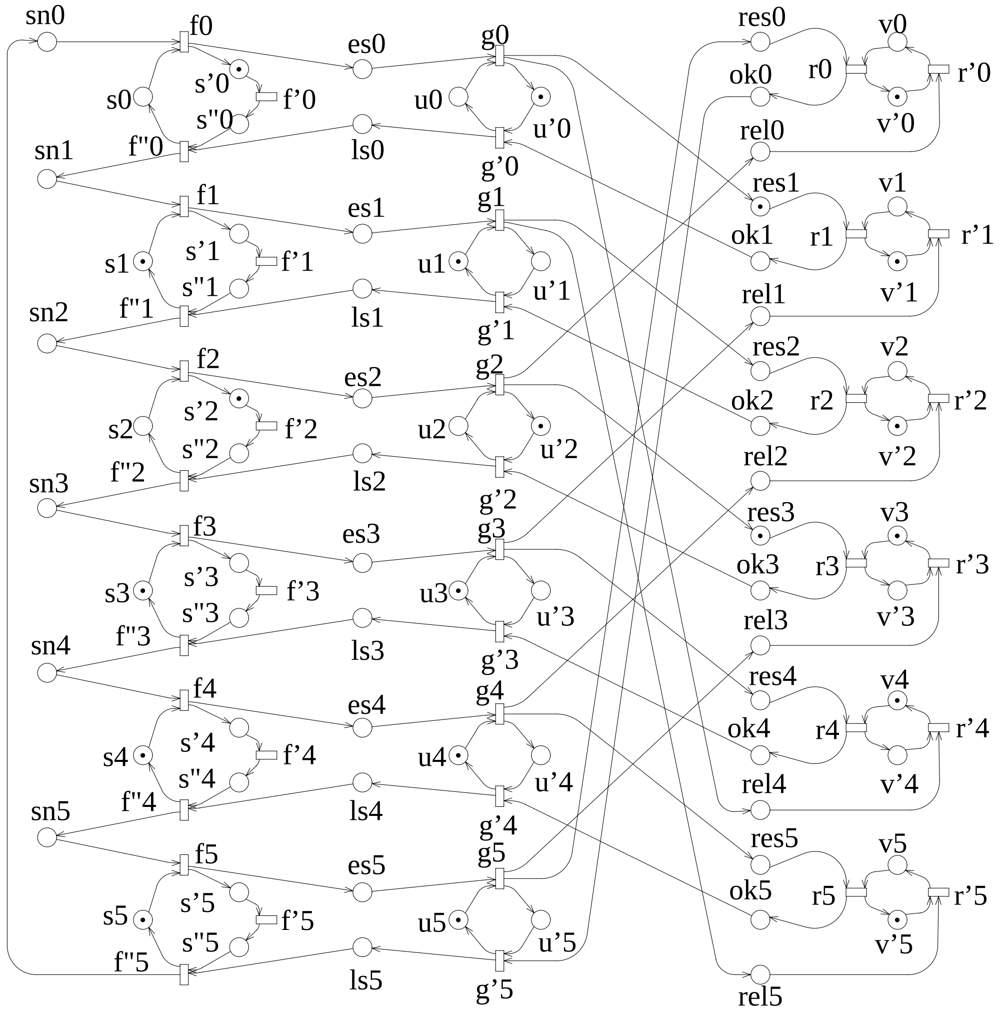

Fig. 5. Petri net model of the train set example

which ratio of the 'physical' and 'electronic' times the cycle time changes significantly.

The numerical results for one and two trains and 6 sections are given in Table 3. As long as λ_{sen} is much smaller (up to three orders of magnitude) than λ_{cross}, the time spent by the controller can be neglected. When λ_{sen} is closer to λ_{cross}, the time used by the controller may become significant, thus impacting the cycle time. From Table 3 it also appears that two trains running in the circuit interfere each other: one train 'locks' the other by forcing it to wait for a section to become free. With six sections, the interference among the two trains increases the cycle time (with respect to one train) of about 20–25%.

The degree of interference seems to depend on the number of sections of a circuit or, conversely, on the number of trains in a given circuit. Thus we computed the cycle time at varying number of sections with two trains (Table 4), and at varying number of trains with 11 sections (Table 5).

Table 3
Cycle time of trains

λ_{sen}	1 train	2 trains	overhead %
1000	600.02	750.02	24.99
100	600.24	750.24	24.99
50	600.48	750.48	24.98
10	602.40	752.43	24.90
5	604.80	754.87	24.81
1	624.00	774.46	24.11
0.5	648.00	799.21	23.33
0.1	840.35	1007.81	19.92
0.075	920.63	1099.68	19.44
0.05	1081.39	1290.54	19.34
0.025	1565.40	1903.33	21.58
0.01	3030.79	3901.04	28.71

Table 4
Cycle time varying the number of sections with two trains

No. of sections	cycle time [s]	optimal value	overhead %
5	667.12	500.40	33.31
6	750.48	600.48	24.98
8	933.94	800.64	16.64
10	1125.75	1000.80	12.48
11	1223.05	1100.88	11.09
12	1320.90	1200.96	9.98

Table 4 reports the optimal values of the cycle time; i.e., considering 1 train on the circuit. It can be seen that the overhead of the measured cycle time with respect to the optimal is decreasing starting from 33.31% (for 5 sections) to 9.98% (for 12). In Table 5, it should be noted that, with increasing number of trains, the dependency makes the cycle time increase over linearly.

Table 5
Cycle time varying the number of trains with 11 sections

No. of trains	cycle time [s]	increment%
1	1100.88	-
2	1223.05	11.09
3	1380.32	12.85
4	1608.87	16.55
5	2049.82	27.40

5.3 Maximum execution time

Using stochastic variables with exponential distribution implies that each transition can experiment an infinite worst case time. Therefore for each path on the net, the worst response time is infinite. In this case, it is interesting to evaluate the time within which the train completes a cycle with a given probability, or, conversely, the probability a cycle will be completed within a given time threshold. Focusing on the former measure, we have derived by simulation the cumulative distribution function of the cycle time, which allows us to compute the time within which the train completes a cycle with 90%, 95%, and 98% probabilities. The distribution functions were represented graphically and the time values corresponding to the given values of the distribution were derived. 1000 measurements were performed and to increase the accuracy, the cycle time was computed at the beginning of each section and then averaged. In this way, for 95% confidence level, 5% confidence interval of the results was assured.

Table 6
Cycle time as a function of λ_{sen}

	$\lambda_{sen}[1/s]$							
	1000	100	50	10	1	0.1	0.025	0.01
average	750.0	750.2	750.5	752.4	774.5	1007.8	1903.3	3901.0
90%	931	931	932	936	951	1165	2188	4396
95%	1050	1053	1057	1064	1096	1293	2388	4811
98%	1215	1218	1221	1226	1238	1440	2594	5187

The time necessary for a train to complete a cycle with 90%, 95%, and 98% probabilities has been computed as a function of three different parameters, namely the speed of the controller, the number of sections and the number of trains. Here for space limitations we show just the results for the first case with 6 sections and 2 trains (Table 6). The results of the measurements are rather regular and consistent with the average times, the maximum time increases in all the three cases with slower sensors, increasing number of sections and increasing number of trains.

5.4 Probability a train has to wait

The system engineer might be interested in the probability that a train has to stop at the end of a section and has to wait for the signal of the controller. By steady state analysis, we computed the distribution of response time of the controller and from it we extracted the probability that a train has to stop at the end of a section and has to wait for the signal of the controller. As before, we considered three different parameters: the speed of controller, the number of sections and the number of trains. The results show a regular behavior and we observed that the probability a train has to wait is more sensitive to the interference among trains than to the other parameters. Thus, the impact of the number of trains is presented in Table 7 with 11 sections and $\lambda_{sen} = 50$. Increasing the number of trains, the probability increases overlinearly, which is in agreement with the overlinear increasing of the cycle time in Table 5.

Table 7
Probability as a function of the number of trains

Number of trains	P(train has to wait)
1	0.000
2	0.111
3	0.250
4	0.432
5	0.699

5.5 Schedulability analysis

All the evaluations shown so far refer to a DFN model in which all the nodes execute possibly in parallel. This way the 'logic' of the design is analyzed and the collected information represents an upper bound of parallelism intrinsic to the design itself. Obviously, to complete the design of a system further steps are required to deal with the supporting hardware architecture. In both cases, when a given architecture is already provided and its usage constitutes a requirement for system development, or a proper hardware architecture must be identified, the activities of the nodes must be associated with processes which may be allocated and scheduled on processors. These steps of design must be analyzed as early as possible to compute the expected timing behavior of the system when the parallel execution is constrained. It may be used, for example, to understand how many processors are necessary such that the response time of the system does not degrade with respect to the ideal case of unlimited parallelism, or to quantify the degradation of the timing behavior of the system, if a given number of processors are available.

Schedulability properties can be easily analyzed in the data flow model. In general, concurrency of data flow nodes can be restricted by introducing the model of a "resource pool", which consists of a channel containing initial tokens representing the resources and two additional nodes collecting and dispatching these tokens. In the case of processor resources, the latter node implements the scheduler. It has the same number of firings as the number of firings of the nodes participated in the scheduling. Assigning priorities to these firings, a non-preemptive, priority based scheduling can be modeled. In nodes which represent activities to be scheduled, additional channels are inserted, to request, get and release resources. A firing of a node, to be enabled, needs the (additional) availability of a resource (processor). Allocation problems can be examined by assigning the nodes to distinguished resource pools (to a set of processors). The integration of the additional nodes and the corresponding modification of the net can be performed automatically.

In our example, the activities of the controller, represented by the firings of CNT_i and RES_i nodes, can be scheduled on one or more processors (the nodes $SECT_i$ represent the environment of the controller, i.e. trains, sensors and actuators). Thus, we measured the average time needed by the controller after receiving the sensor signal to activate the actuator signal of a section (let a given train move to the next section). To do this, the controller was analyzed modeling the arrival of sensor signals and the movement of trains by immediate firings. The measurement results for 9 sections and $\lambda_{cnt} = \lambda_{res} = \lambda_{rel} = 500$ (corresponding to the previously used $\lambda_{sen} = 50$ value) are presented in Table 8.

Table 8
Average response time of the controller [msec]

Number of processors	1 train	2 trains	3 trains	4 trains
1	8.0	16.0	24.0	32.0
2	6.1	8.1	12.1	32.0
3	6.0	6.9	9.9	32.0
4	6.0	6.8	9.9	32.0

In case of 1 train, the reservation of a section and the release of the other one (left by the train) can be done in parallel, this way the application of two processors can speed up the controller. In case of 2 or 3 trains, 3 processors are needed to avoid considerable time overhead. If 4 trains are in the system, there are few parallel activities of the controller because the trains have to wait for each other, thus the response time does not depend on the number of processors.

5.6 Cost of the analysis

The time and hardware resources needed for the measurements consist of those necessary for the transformation and those required by the simulation and analysis tools. The automatical data flow network to Petri net transformation is a linear algorithm (with respect to the number of firings in the DFN model), and it can be performed considerably fast even on common PCs (in about a few seconds for the networks analyzed here). Considering the simulation and analytical solution of the resulting Petri nets, we utilized (by the transformation) well known analysis and simulation packages, and did not introduce new techniques. Thus, the required resources and limitations are that of the underlying tools.

The analytical solution of the Petri net by the SPNP tool includes the generation of the reachability graph (RG) of the model, resulting in extensive memory allocation, which restricts the analysis to workstations with paging and virtual memory capabilities. The size of the RG can not be expressed in terms of the number of nodes in the network, and in our examples depends also on the number of trains. In our experiments the average size was of few megabytes with a maximum of about 20 megabytes. Even in this case the generation of the RG and the solution of the net required less than 10 minutes on a Sun Sparc 10 workstation.

Using simulation tools, the main requirement is the time needed to perform the simulation. The SimNet package running on a 66MHz PC needed about 15 minutes to perform 1000 measurements on the largest net (11 sections, 5 trains). More sophisticated simulation tools running on fast workstations can reduce this time significantly.

6 Concluding remarks

In this paper we have defined a formal model of asynchronous uninterpreted data flow networks to be used as a framework for the design process of those control systems that rely for their behavior just on the presence or absence of signals. To represent the state-dependent behavior of the system, we have introduced *states* of the nodes. To represent priority constraints, *priorities* are assigned to the activities of the nodes. Additionally, to be able to investigate the timing properties, the activities are associated with a *timing parameter* that denotes a delay for the execution of the activity.

A data flow network is usually very close to the intuitive representation of the specification of a control system, as conceived by a control engineer and, fitting the designers' way of thinking, allows to reduce the time to construct the model. Due to the extensions introduced, our approach unifies the advantages of strict data flow models and FSM description techniques. Beside the intuitive description, the possibility to assign formal semantics enables the formal verification of data flow modeled systems.

The lack of a widely accepted model of data flow networks resulted in the shortage of automatic tools which could support the evaluation of the model. To be able to analyze the properties of the design, we have followed a rather common approach used for almost all high level formalisms such as high level Petri nets. In these formalisms an indirect evaluation is made possible by resorting to the tools available for lower level representations (as most of the analysis on Petri nets are performed through Markov models). Our formalism, as well as high level and hierarchical Petri nets, resort to plain Petri net tools for analysis. Automatic model transformations and the back annotation of the results into the original model will hide the specific representations and tools required for the validation.

In this work we defined a transformation towards Timed Transition Petri nets and proved that it preserves the timing properties of the data flow network. This transformation can be performed automatically and enables the use of simulation and analytical tools available for TTPN.

The application of this method to an example, even though a simple one, has been conducted referring to the DSPN subclass of TTPN and allowing to appreciate the kind of analysis which can be automatically performed. The choice of DSPN allows to obtain analytical solutions of the network restricted to deterministic and exponentially distributed firing delays, still admitting simulation for more general cases. However, it has to be emphasized that the transformation is valid for the entire TTPN class, thus including all data flow networks whose timing variables follow different distributions. As soon as new tools dealing with non-exponential distributions will be available they can be directly used by applying our transformation.

References

[1] B. Antal, User's Manual for the *df2pn* Package. *University of Pisa, Italy*, 1995

[2] C. Bernardeschi, A. Bondavalli and L. Simoncini, Data Flow control systems:

an example of safety validation, *Proceedings SAFECOMP'93* (Poznan, Poland, 1993) 9–20.

[3] *Proceedings International Workshop on Timed Petri Nets*, IEE CS press, (Torino, Italy, 1985).

[4] A. Bondavalli and L. Simoncini, Functional paradigm for designing dependable large-scale parallel computing systems, *Proceedings of the International Symposium on Autonomous Decentralized Systems, ISADS '93* (Kawasaki, Japan, 1993) 108–114.

[5] A. Bondavalli, L. Strigini and L. Simoncini, Data Flow-like languages for real-time systems: Issues of computational models and notations, *Proceedings of the International Symposium on Reliable Distributed Systems, SRDS-11* (Houston, Texas, 1992).

[6] R. Braek and O. Haugen, Engineering Real Time Systems. (Prentice Hall, 1993)

[7] J. T. Buck, S. Ha, E. A. Lee and D. G. Messerschmitt, Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *Int. Journal of Computer Simulation*, vol. 4, April 1994, 155–182.

[8] A. Bondavalli, A. Buzzi and F. Tarini, Uno strumento grafico per la strutturazione di applicazioni tolleranti i guasti, *Proc. Congresso annuale A.I.C.A. 95* (Cagliari, Italy, 1995) 979-986.

[9] Gy. Csertán, C. Bernardeschi, A. Bondavalli and L. Simoncini. Analysis of temporal properties of data flow control systems. *Proceedings 12th IFAC Workshop on Distributed Computer Control Systems, DCCS-94*, (Toledo, Spain, 1994) 153–158.

[10] Gy. Csertán, A. Pataricza and E. Selényi, Dependability analysis in hw-sw co-design. *Proc. of the IEEE International Computer Performance and Dependability Symposium IPDS'95* (Erlangen, Germany, 1995) 316–325.

[11] P. Le Guernic, T. Gautier, M. Le Borgne and C. Le Maire, Programming real-time applications with Signal, *IEEE Proceedings* (1991) 1321–1336.

[12] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, The synchronous data flow programming language LUSTRE, *IEEE Proceedings* (1991) 1305–1320.

[13] D. J. Hatley and I. A. Pirbhai, Strategies for Real-Time System Specification. (Dorset House, 1987)

[14] M. A. Holliday and M. K. Vernon, A generalised timed Petri net model for performance analysis, *Proceedings Workshop on Timed Petri Nets*, IEEE (Torino, Italy, 1985).

[15] , K. Jensen, Coloured Petri Nets: Basic concepts, analysis methods and practical use, Vol. I: Basic concepts. *Monographs in theoretical computer science*, (Springer Verlag, 1992)

[16] K. Jensen and G. Rozenberg (eds), High Level Petri Nets. Theory and Application. (Springer Verlag, 1991)

[17] B. Jonsson, A fully abstract trace model for data flow networks, *Proceedings of the 16th ACM symposium on POPL* (Austin, Texas, 1989) 155–165.

[18] G. Kahn, The semantics of a simple language for parallel programming, *Proceedings of the IFIP '74* (North Holland, 1974) 471–475.

[19] A. Kalavade and E. A. Lee, A Hardware-Software Codesign Methodology for DSP Applications, *IEEE Design & Test of Computers*, **3** (1993) 16–28.

[20] K. M. Kavi, B. P. Buckles and U. N. Bhat, Isomorphism between Petri nets and data flow graphs, *IEEE Transactions on Software Engineering*, **13**, 10 (1987) 1127–1134.

[21] H. Kopetz, Time-Triggered vs Event Triggered real-time systems, *Proceedings Workshop Operating Systems of the 90ties and beyond*, Springer-Verlag (1991).

[22] G. Le Lann, Designing real-time dependable distributed systems, *Internal Report* **1425**, INRIA (Rocquencourt, 1991).

[23] G. Ciardo, J. Muppala and K. Trivedi, SPNP: Stochastic Petri Net Package. *Int. Conf. on Petri Nets and Performance Models*, (Kyoto, Japan, December 1989)

[24] M. Ajmone Marsan and G. Chiola, On Petri nets with deterministic and exponentially distributed firing times, in: G. Rozenberg, editor, *Advances in Petri Nets 1987, Lecture Notes on Computer Science*, **266**, Springer Verlag (1987) 132–145.

[25] B. Lent and H. Kurmann, The OR data flow Architecture for a Machine Embedded Control System, *J. Real-Time Systems*, **1** (1989) 107-132.

[26] I. Majzik, On semantics and temporal analysis of data flow networks, *Internal report, Department of Information Engineering, University of Pisa* (Pisa, Italy, 1994).

[27] R. Jagannathan and E. A. Ashcroft, Fault Tolerance in Parallel Implementations of Functional Languages, *Proceedings FTCS-21* (Montreal, 1991) 256-263.

[28] A. Saed, R. de Lemos and T. Anderson, The role of formal methods in the requirements analysis of safety-critical systems: A train set example, *Proceedings FTCS-21* (Montreal, Canada, 1991) 478–485.

[29] M. Takesue, Data Flow Computer Extension towards Real-Time Processing, *J. Real-Time Systems*, **1** (1989) 333–350.

[30] P. Ward and S. Mellor, Structured Development for Real-Time Systems. (Prentice Hall, 1986)