ANALYSIS OF TEMPORAL PROPERTIES OF DATA FLOW CONTROL SYSTEMS

GY. CSERTÁN[*], C. BERNADESCHI[†], A. BONDAVALLI[‡] and L. SIMONCINI[†]

[*] *Technical University of Budapest, Department of Measurement and Instrument Engineering, Műegyetem rkp. 9, H-1521 Budapest, Hungary*
[†] *Universita di Pisa, Dipartimento di Ingegneria dell'Informazione, Via Diotisalvi 2, I-56100 Pisa, Italy*
[‡] *CNUCE-CNR, Via S. Maria 36, I-56126 Pisa, Italy*

Abstract. This paper investigates the analysis of temporal properties of control systems modelled using the data flow computational paradigm. A transformation from data flow networks to timed Petri nets is defined. It preserves temporal properties and allows, through the analysis of the Petri net, the indirect evaluation of the properties of the data flow network. The paper contains an example for explaining the transformation and showing which kind of analyses can be performed.

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

1. INTRODUCTION

Early timing analysis may be very important for the development process of control systems. 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. 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. Clearly, depending on the kind of system at hand, the quantities of interest are rather different. In trying to demonstrate that a given design of a real-time control system satisfies the timing requirements, the maximum execution and/or response time must be provided, which must be computed along all possible execution paths and possible system states. On the other side, if the purpose is just to predict the expected time performance of a system, the average response time, the average execution time and the steady-state analyses are of interest.

Due to their distributed/parallel and data-driven nature, control systems can be easily modelled by data flow networks. Data flow models have the advantages of a simple graphical representation (data flow graphs), compactness, expressiveness of the parallelism inherent in the modelled system and others (Bondavalli *et al.*, 1992; Jagannathan and Ashcroft, 1991). Moreover, it is interesting to note that data flow concepts have been considered as an appropriate means of organising real-time processing (Lent and Kurmann, 1989; Takesue, 1990). Unfortunately however, data flow models lack of methods and automatic tools for analysing their properties. On the contrary, using directly other formalisms, like Petri nets, for which a lot of analysis tools are available, has the disadvantage of needing to cope with very large and complex models, not always well dominated by the designer.

This paper investigates the analysis of temporal properties of control systems modelled using the data flow paradigm. A transformation is defined from data flow networks to timed Petri nets, which are known for modelling very well concurrent, deterministic and stochastic systems. From the point of view of temporal behaviour, the transformation is proved to generate an isomorphic Petri net. Therefore, it permits, through the analysis of the Petri net, the indirect evaluation of the data flow model. Due to space restrictions, the transformation, which is presented in (Csertán, 1993), is not described in details; an example is used instead having also the purpose of showing which kind of analyses can be performed.

The rest of the paper is organised as follows. Section 2 contains first a description of our data flow model, which includes timing information, and addresses the temporal properties of a control system which may be of interest and may be derived in this framework.

Section 3 introduces the Petri nets proposed for dealing with time and gives hints on the transformation. Section 4 is devoted to the example of a train set. Starting from the specification, the data flow design is shown and the resulting Petri net is then derived and evaluated. This application does not include timing constraints, therefore the kind of evaluation carried out regards just performance issues. Finally Section 5 contains some concluding remarks.

2. DATA FLOW NETWORKS FOR CONTROL SYSTEMS

In (Bondavalli and Simoncini, 1993; Bernardeschi *et al.*, 1993) a data flow computational model is proposed for allowing early analyses of control systems. In this model, the control system corresponds to a data flow network whose input and output events describe the interaction between the control system and its environment. The control system is made up of sensors, actuators and a controller. The controller executes the control algorithm, processing the parameters of the environment sent as signals by the sensors. According to the results of the computation, the controller sends signals to the actuators to intervene in the environment.

The external environment can be modelled together with the controller to obtain a closed network, thus allowing validation and evaluation of properties by means of analytical models. The controlled system is specified at a very high abstraction level, by considering the load of the controlled system and input/output signals generated accordingly. In this setting, the user describes the behaviour of the controlled system by defining the distribution of the sensor and actuator signals. On open data flow networks, instead, only simulation can be executed to check properties.

A data flow network N is a set of nodes P_N, which execute concurrently and exchange data over one-to-one FIFO channels. The functional behaviour of a node is given by a set of firings (behaviours); a node is ready to execute as soon as the data required by one of its firings are available. In addition to this basic functionality of a node, timing characteristics of the computing nodes are taken into account by associating to each firing of a data flow node the time it takes to be executed. A priority is also associated to each firing of a node. For each node, when more firings are verified by the presence of data over channels, the one with the greatest priority is enabled and selected for execution. Assigning different priorities to each firing of a node admits a nondeterministic behaviour based on the presence/absence of data, while, given a configuration of the data over the channels, it constrains the behaviour of the node to be deterministic.

Definition 1 *A node p is a tuple $p = (I_p, O_p, S_p, R_p, \Pi_p, \lambda_p)$ where:*

I_p - *set of input channels*
O_p - *set of output channels*
S_p - *set of states,* $s_p^0 \in S_p$ - *initial state*
R_p - *set of firings*
where $f \in R_p$ *is a tuple* (s, X_{in}, s', X_{out})
$s, s' \in S_p$ - *states before and after the firing*
$X_{in} : I_p \mapsto \mathbb{N}$ - *input tokens*
$X_{out} : O_p \mapsto \mathbb{N}$ - *output tokens*
$\Pi_p : R_p \mapsto \mathbb{N}$ - *priority function*
$\lambda_p : R_p \mapsto \{\mathbb{R}^+ \cup \{0\}\}$ - *time function*

The meaning of $f = (s, X_{in}, s', X_{out})$ is that if the node is in state s, each input channel $i \in I_p$ holds at least $X_{in}(i)$ tokens, and no other firings are enabled being on higher priority level than $\Pi_p(f)$, then firing f is selected for execution. The execution of the firing removes $X_{in}(i)$ tokens from each input channel $i \in I_p$ and outputs $X_{out}(j)$ tokens on each output channel $j \in O_p$, while the node p changes its state from s to s'. The firing takes $\lambda_p(f)$ time to be executed. During execution of a firing the node is in working state, $s_p^w \in S_p$.

The channels of a data flow network N may link two nodes (internal channels) or be connected to just one node (input/output channels) to represent interactions between the control system and its environment in case of open networks. Communication events occur when tokens are inserted into an input channel (input event) or tokens are removed from an output channel of the network (output event). A network transition can be generated by the firing of a node or by a communication event.

Definition 2 *The data flow network N composed by the set P_N of nodes, is defined by:*

$C_N = \bigcup_{p \in P_N} (I_p \cup O_p)$ - *set of channels*
$I_N = (\bigcup_{p \in P_N} I_p) \setminus (\bigcup_{p \in P_N} O_p)$ - *input channels*
$O_N = (\bigcup_{p \in P_N} O_p) \setminus (\bigcup_{p \in P_N} I_p)$ - *output channels*
$R_N = R_{in} \cup R_{out} \cup R_{int}$ - *set of events*
R_{in} - *set of input events*
and $\lambda : R_{in} \mapsto \{\mathbb{R} \cup \{0\}\}$ *is its time function*
R_{out} - *set of output events*
and $\lambda : R_{out} \mapsto \{\mathbb{R} \cup \{0\}\}$ *is its time function*
$R_{int} = \bigcup_{p \in P_N} R_p$ - *set of internal events*
$\Sigma_N = \Sigma_{C_N} \otimes \Sigma_{P_N}$ - *set of states, where*
$\otimes$ *denotes the Cartesian product,*
σ^0 - *the initial state*
$\Sigma_{C_N} : C_N \mapsto \mathbb{N}$ - *state of channels*
$\Sigma_{P_N}(p) \in S_p, \forall p \in P_N$ - *state of nodes*

An input event $r \in R_{in}$ finishes the execution when its firing time expires and starts a new execution immediately. An output event $r \in R_{out}$ starts the execution upon arrival of tokens to the corresponding output channel and finishes it when the firing time expires. An internal event $r \in R_{int}$, which corresponds to a firing of a node, starts the execution when it becomes enabled and finishes after expiring of the firing time. A parallel execution of all the selected firings (at most one for each node) is therefore possible, and will actually be performed in an implementation according to the available computational resources. The analyses will be performed considering this extreme level of parallelism where no delays are added due to lack of resources so providing an upper bound on the ideal timing properties admitted by the design.

One of the most important characteristics of a real-time control system is the maximum response time, which is the time value elapsed between the arrival of a signal from the environment and the sending of the corresponding command to the actuator. The maximum is computed over all possible system activities under any circumstances, i.e. no matter in which state the system was when the input signal was received or which other activities were executed concurrently. By computing an average instead of the maximum the average execution time also called average response time is obtained. The analysis methods applied to study timing properties are: transient analysis and steady-state analysis. One of the possible implementations of transient analysis is to start the control system from a given state and to let the controller execute as far as possible (i.e. by starting all possible activities). This kind of analysis does not allow to receive input signals and the controller is isolated from the environment.

Execution time of activities is supposed to be a known exponentially distributed, stochastic variable or a known fixed, deterministic value. The exponential distribution refers to the fact, that during normal operation (high probability) a component is supposed to have an execution time with lower and upper bounds, while a faulty component (low probability) may have very large even infinite execution time. In this case, obviously, the maximum execution time of any system is infinite. Still one can try to give a probabilistic timing assessment: find a time threshold τ such that the execution will terminate by τ with the desired (high) probability.

3. FROM DATA FLOW NETS TO PETRI NETS

A Petri net is a bipartite graph with two types of nodes; places and transitions. Places may contain tokens, and the current distribution of tokens over the places denotes the state of the modelled system. On the other hand places represent the conditions (pre- and post conditions) to allow a transition to execute. The execution of a transition changes the distribution of tokens and thus represents the state change (event) of the system under study. Timed Petri nets were introduced by extending the original formalism with the notion of time, where time parameter can be assigned to transitions or to places. Mainly due to theoretical problems for the case where time is associated to places no analysis tools have been developed. For the other case, in which timing parameter can be interpreted as execution time of events, a rich set of tools and methods is available. As theoretical background they usually adopt Markov chains.

In (Csertán, 1993) a transformation from data flow networks to timed Petri nets is defined. After extensive studies of many different types of timed Petri nets, for which automatic tools are available to support the analysis of the network, the class of Deterministic and Stochastic Petri nets (DSPN) has been chosen as a target model (Ajmone Marsan and Chiola, 1987). Each channel of the data flow network and each state of a node is simply mapped into a place of the Petri net, while each firing is mapped into two transitions: an immediate transition, which denotes the starting phase of the firing, and holds the priority property of the firing, and a timed transition, which inherits the timing properties of the firing. Arcs of the net correspond to the links of the data flow network, arc weights are set according to the input and output mappings of firings. From the point of view of temporal behaviour, the transformation is proved to generate an isomorphic Petri net.

4. AN EXAMPLE OF TIMING ANALYSIS

The train set example described in (Saed *et al.*, 1991), where trains move unidirectionally along a circuit, (see Fig. 1) is now considered. The time parameters have been chosen to be exponentially distributed stochastic variables. For this example, the data flow specification will be given, the corresponding Petri net derived and analysed using the *GreatSPN* tool (Chiola, 1987). Results regarding the steady-state analysis are here reported.

With the assumption that the train's length is less than each section's length, a safety criterion states 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.

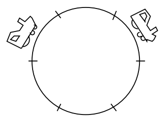

Fig 1. The train set example

According to the proposed modelling approach, the system is divided into two subparts, the model of the plant and the model of the controller connected by sensor and actuator signals, as shown by Fig. 2 in the case of two trains and six 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 the modulo–6 addition and subtraction, respectively). If the reservation is successful CNT_i sends the `ls` signal to the section. The second part of the controller, nodes RES_i, behaves like a memory keeping track the reserved and

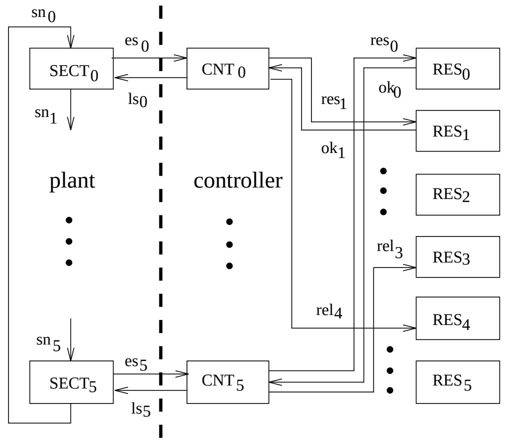

Fig 2. Data flow model of the train set example

free sections. Receiving a `rel` signal it releases the section, receiving a `res` signal it reserves the section. Of course if a given section is reserved for a train it can not be reserved for another one. The two trains are supposed to be in sections $SECT_0$ and $SECT_2$ initially.

The timing variables of the example (all exponentially distributed) are the following:

- τ_{sen} - time consumed by a sensor sending a signal (with parameter λ_{sen});
- τ_{cross} - time a train needs to move along the section (λ_{cross});
- τ_{act} - time spent by receiving an actuator signal (λ_{act});
- τ_{cnt} - time the controller needs to send signals (λ_{cnt});
- τ_{res} - time for reserving a section (λ_{res});
- τ_{rel} - time for releasing a section (λ_{rel}).

The resulting data flow specification is:

$$
\begin{aligned}
&N = \bigcup_{i=0}^{5} \{SECT_i, CNT_i, RES_i\}\\
&\Sigma_C^0 : \forall i, sn_i, es_i, ls_i, ok_i, rel_i \mapsto 0\\
&\quad res_1 \mapsto 1,\ res_3 \mapsto 1,\ res_{\{0,2,4,5\}} \mapsto 0\\
&\Sigma_{P_N}^0(p) = s_p^0, \forall p \in P_N\\
&SECT_i :\\
&\quad I_{SECT_i} = \{sn_i, ls_i\} \qquad O_{SECT_i} = \{sn_{i\oplus 1}, es_i\}\\
&\quad S_{SECT_i} = \{s_i, s_i', s_i''\} \qquad s_{SECT_i}^0 = s_i, i = 1,3,4,5\\
&\qquad\qquad\qquad\qquad\qquad s_{SECT_i}^0 = s_i', i = 0,2\\
&\quad R_{SECT_i} = \{f = (s_i, [sn_i \rightarrow 1], s_i', [es_i \rightarrow 1]),\\
&\qquad\qquad f' = (s_i', [], s_i'', []),\\
&\qquad\qquad f'' = (s_i'', [ls_i \rightarrow 1], s_i, [sn_{i\oplus 1} \rightarrow 1])\}\\
&\quad \Pi_{SECT_i}(f) = 0, \forall f \qquad \lambda_{SECT_i}(f) = \lambda_{sen}\\
&\quad \lambda_{SECT_i}(f') = \lambda_{cross} \qquad \lambda_{SECT_i}(f'') = \lambda_{act}\\
&CNT_i :\\
&\quad I_{CNT_i} = \{es_i, ok_i\} \qquad O_{CNT_i} = \{ls_i, res_i, rel_{i\ominus 2}\}\\
&\quad S_{CNT_i} = \{s_i, s_i'\} \qquad s_{CNT_i}^0 = s_i, i = 1,3,4,5\\
&\qquad\qquad\qquad\qquad\qquad s_{CNT_i}^0 = s_i', i = 0,2\\
&\quad R_{CNT_i} = \{f = (s_i', [ok_{i\oplus 1} \rightarrow 1], s_i, [ls_i \rightarrow 1]),\\
&\quad f' = (s_i, [es_i \rightarrow 1], s_i', [res_{i\oplus 1} \rightarrow 1, rel_{i\ominus 2} \rightarrow 1])\}\\
&\quad \Pi_{CNT_i}(f) = 0, \forall f \qquad \lambda_{CNT_i}(f) = \lambda_{cnt}, \forall f\\
&RES_i :\\
&\quad I_{RES_i} = \{res_i, rel_i\} \qquad O_{RES_i} = \{ok_i\}\\
&\quad S_{RES_i} = \{s_i, s_i'\} \qquad s_{CNT_i}^0 = s_i, i = 3,4\\
&\qquad\qquad\qquad\qquad\qquad s_{CNT_i}^0 = s_i', i = 0,1,2,5\\
&\quad R_{RES_i} = \{f = (s_i, [res_i \rightarrow 1], s_i', [ok_i \rightarrow 1]),\\
&\qquad\qquad f' = (s_i', [rel_i \rightarrow 1], s_i, [])\}\\
&\quad \Pi_{RES_i}(f) = 0, \forall f \qquad \lambda_{RES_i}(f) = \lambda_{res}\\
&\qquad\qquad\qquad\qquad\qquad \lambda_{RES_i}(f') = \lambda_{rel}
\end{aligned}
$$

4.1 The Petri Net Equivalent to the Data Flow Net

The Petri net derived by applying the transformation is depicted in Fig. 3. Places corresponding to the channels are referred to with the same name, while those denoting the internal state of data flow nodes and transitions are numbered increasingly. Immediate transitions and additional places are omitted to keep the Petri net as simple as possible.

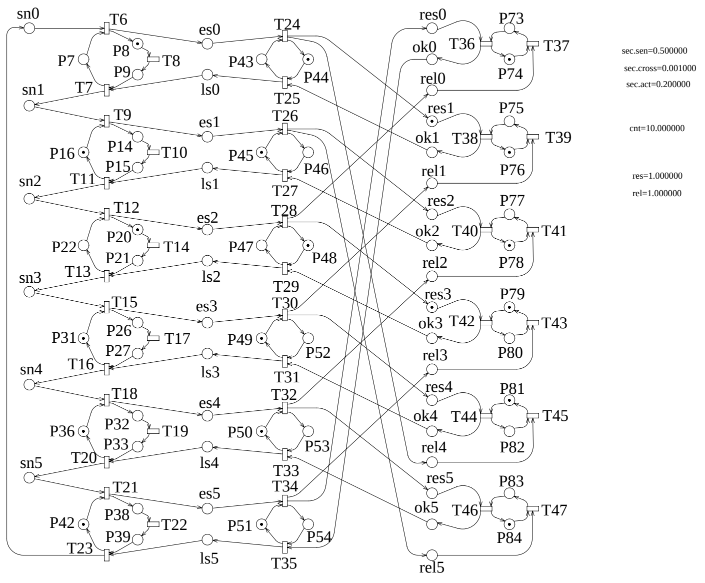

Fig 3. Petri net model of the train set example

Transitions T6, T9, T12, T15, T18, T21 symbolise the sending of a sensor signal when a train has entered the section. Firing of transitions T8, T10, T14, T17, T19, T22 represents the movement of a train from the beginning of the section to its end and T7, T11, T13, T16, T20, T23 the reception of the actuator signal. T24, T26, T28, T30, T32, T34 correspond to receiving the sensor signal and starting the reservation and release of sections, while T25, T27, T29, T31, T33, T35 correspond to sending the actuator signal to the sections thereby allowing trains to proceed. Finally T36, T38, T40, T42, T44, T46 represent the reservation of sections and T37, T39, T41, T43, T45, T47 their release. Before leaving some section and until entering the next one, a train has three sections reserved. A train in $SECT_3$ has $SECT_3$ and $SECT_2$ reserved, it can move forward only after $SECT_4$ has been reserved too, this happens with the firing of transition T4. All the three sections remain reserved until firing of T41, which releases $SECT_2$. T41 may be fired only after the train has entered $SECT_4$ (transition T18).

4.2 Analysis of the Example

Since this is not a real time application, the average time a train needs to cover the whole circuit is of main concern. To this purpose, steady-state analysis is executed, thus obtaining the reachability set of the Petri net. The results of steady-state analysis, done by *GreatSPN*, give 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 transition T6 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 T6 will fire once for each train. Therefore the cycle time is: $\tau_{cycle} = n/throughput(T6)$ 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.

Table 1: Parameters and their value

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

The first analysis performed aims at verifying that, when the time necessary to trains to cross sections (λ_{cross}) is several orders of magnitude bigger 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. Sections are supposed to be long about 1.5 Km and $\lambda_{cross} = 0.01$ gives an average time of 100 seconds for a train to cross it. The values for the other parameters are reported in Table 1, where λ_{sen} has been selected to range between 1000 and 0.01 (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 which ratio of the 'physical' and 'electronic' times the cycle time changes significantly.

The numerical results for one and two trains are given in Table 2, while Fig. 4 shows the results in a diagram. 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. The cycle time in case of one train is very close to $6 * 1/\lambda_{cross}$. When λ_{sen} is closer to λ_{cross}, the time used by the controller may become significant, thus impacting the cycle time (right end of Fig. 4). The same observation holds in case of two trains. If two trains are in the circuit, they interfere each other: one train 'locks' the other by forcing it to wait for a section to become free. The interference between trains remains unaltered until the time necessary to the controller becomes relevant for the cycle time. In this experiment, with two trains and six sections, the interference increases the cycle time with respect to one train by a factor of 1.25.

Table 2: Cycle time of trains

λ_{sen}	cycle time (1)	cycle time (2)
1000	600.02	750.02
100	600.24	750.24
50	600.48	750.48
10	602.40	752.43
5	604.80	754.87
1	624.00	774.46
0.5	648.00	799.21
0.1	840.35	1007.81
0.075	920.63	1099.68
0.05	1081.39	1290.54
0.025	1565.40	1903.33
0.01	3030.79	3901.04

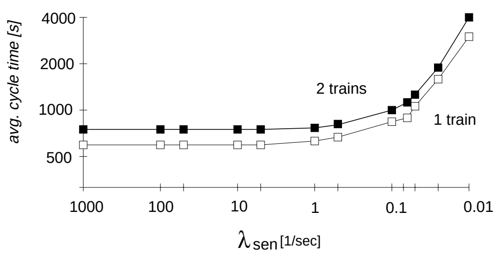

Fig 4. Cycle time of trains

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. Other two settings have been used to check this conjecture. $\lambda_{sen} = 50$, a reasonable value, that gives the average sensor time of 0.02sec and the average controller time of 0.006sec, has been set leaving the other parameters unaltered. The cycle time has been computed for two trains running on circuits of $5, 6, 8, 10, 11, 12$ sections and in a circuit of 11 sections the cycle time of $1, 2, 3, 4, 5$ trains. The results of varying the number of sections are in Table 3 graphically represented in Fig. 5, while the cycle time of varying the number of trains is reported in Table 4 and Fig. 6.

Table 3: Cycle time varying the number of sections

No. of sections	cycle time [s]
5	667.12
6	750.48
8	933.94
10	1125.75
11	1223.05
12	1320.90

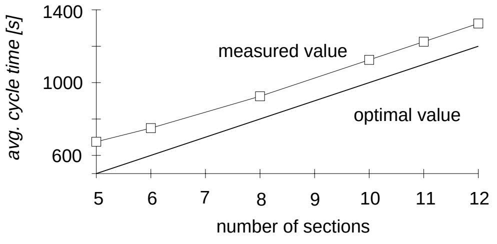

Fig 5. Cycle time varying the number of sections

The continuous line in Fig. 5 shows the theoretical optimal values of the cycle time; i.e., considering 1 train on the circuit and an instantaneous controller. It can be seen that the ratio between the measured cycle time and the ideal optimal is decreasing starting from 1.334 (for 5 sections) to 1.1 (for 12). The same trend can be observed from Fig. 6, where it is clearly seen that, with increasing number of trains, the dependency makes the cycle time increase over linearly.

Table 4: Cycle time varying the number of trains

No. of trains	cycle time [s]
1	1100.88
2	1223.05
3	1380.32
4	1608.87
5	2049.82

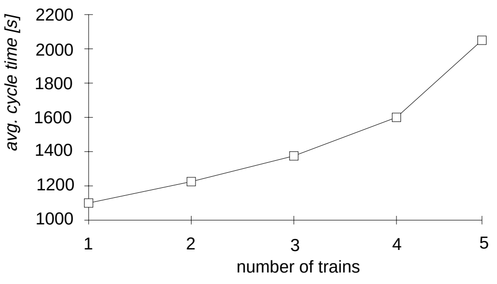

Fig 6. Cycle time varying the number of trains

5. CONCLUDING REMARKS

In this paper the analysis of temporal properties of control systems modelled using a data flow computational paradigm has been addressed. The analysis of the Petri net obtained by a transformation preserving temporal properties has been used for the evaluation of data flow networks. An example is described related to a non real-time application. The analysis performed aimed to quantify the average cycle time in different contexts to measure the impact of the number of trains or length of the circuit on the overall performance as well as the interference between trains when they become too close to each other.

In some contexts, it may be interesting to run a transient analysis measuring the time between two events, e.g. the time necessary to execute some part of the controller. In order to measure maximum times, necessary for analysing real time applications, two possible approaches are practicable. Either one can use different distributions that have a maximum for each transition like the uniform one or compute the maximum as a value such that the execution will terminate with the desired (high) probability. In the former case the GreatSPN tool allows just deterministic distribution so some extension should be developed. The latter case requires a very careful management of probabilities. Future work will focus on possible improvements of the transformation in order to allow a more detailed temporal analysis. A refinement of the timing parameters to reflect more lifelike distributions will also be considered.

6. REFERENCES

Ajmone Marsan, M. and G. Chiola (1987). On petri nets with deterministic and exponentially distributed firing times. *Proc. Advances in Petri Nets 1987*, LNCS **266**, 132–145, Springer-Verlag.

Bernardeschi, C., A. Bondavalli and L. Simoncini (1993). Data flow control systems: an example of safety validation. *Proc. SAFECOMP'93*, Poznan, Poland, 9–20, Springer-Verlag.

Bondavalli, A., L. Strigini and L. Simoncini (1992). Data-Flow like Languages for Real-Time Systems: Issues of Computational Models and Notation. *Proc. 11-th Symposium on Reliable Distributed Systems*, Houston, USA, 214–221.

Bondavalli, A., and L. Simoncini (1993). Functional Paradigm for Designing Dependable Large-Scale Parallel Computing Systems. *Proc. ISADS '93 Int. Symp. on Autonomous Decentralized Systems*, Kawasaki, Japan, 108–114.

Chiola G. (1987). A graphical net tool for performance analysis. *Proc. 3rd Int. Workshop on Modeling Techniques and Performance Evaluation*, Paris, France.

Csertán Gy. (1993). Temporal Analysis of data flow computational paradigm based on control Systems. *Internal Report IR-1/93-LS-CsGy*, Department of Information Engineering, University of Pisa.

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

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

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

Takesue M. (1990). Dataflow Computer Extension towards Real-Time Processing. *International Journal of Real-Time Systems*, **1**, 333-350.