Reachability and Timing Analysis in Data Flow Networks: A Case Study

B. Antal, Gy. Csertán and I. Majzik
Technical University of Budapest
Dept. of Measurement and Instr. Eng.
Budapest, H-1521, Hungary
majzik@mmt.bme.hu

A. Bondavalli and L. Simoncini
CNUCE-CNR
Pisa, I-56100, Italy
Via S. Maria 36
a.bondavalli@cnuce.cnr.it

Abstract

The need of efficient implementation, safety and performance requires early validation in the design of computer control systems. The detailed timing and reachability analysis in the development process is particularly important if we design equipments or algorithms of high performance and availability. In this paper we present a case study related to the early validation of control systems modeled by data flow networks. The model is validated indirectly as it is transformed to Petri nets in order to be able to utilize the tools available for Petri nets.

1. Introduction

Compactness, modularity, data driven and distributed nature, the support of stepwise refinement and direct implementation makes the data flow modeling paradigm attractive for system designers [3]. A current disadvantage of the approach is that the lack of common semantics results in the shortage of available analysis and evaluation tools. Performance and safety analysis of data flow modeled systems can not be performed easily.

The problem can be solved by indirect analysis of data flow networks (DFN). A set of model transformations exist which preserve selected properties of the model and result in a representation which can be analyzed by a variety of sophisticated tools. The formal background and (theoretical) proof of isomorphism regarding the properties to be analyzed assures that the results gained by the analysis of the transformed model can be propagated back to the original data flow representation.

In this paper, we show that reachability, early performance and timing analysis of data flow modeled systems can be performed through an automatic data flow net to Petri net transformation. This way, the wide range of tools available for Petri nets can be utilized without coping with construction of large and complex Petri net models, which is done automatically.

The correspondence between data flow networks and Petri nets is a natural idea [6]. However, the proof of isomorphism first requires to define the type and exact semantics of data flow networks and then the class of the corresponding Petri nets. [5] introduces a transformation from DFN models to timed Petri nets of the class Deterministic and Stochastic Petri Nets [8]. [7] proves that the transformation preserves timing and reachability properties, thus the analysis of data flow networks can be performed on the equivalent Petri net.

The formal and theoretical background of the model transformation is a topic of other papers, here we concentrate on presenting the usefulness of the transformation by an application example. As part of the model of a complex computer system, the (sub)model of a disk handler is presented. We selected this example since it has a compact representation describing hardware as well as software parts of the system, it is easy to survey and despite of the simple representation useful measures of the system can be derived. Our goal is to show the methodology of the network analysis without dealing with the overhead caused by a more complex model.

The rest of the paper is organized as follows. In Section 2 the modeling approach is presented. First our data flow notation is defined which is intended to model distributed control systems. The Petri net notation and the corresponding analysis tools are also recalled. The model transformation is introduced in Section 2.4, the equivalence of the two representations is discussed in Section 2.5. The results of the model analysis are presented in Sections 3 and 4. The paper is closed by a short conclusion.

2. Modeling approach

The early validation of control systems is regularly performed at higher level of model abstraction where the designer does not want to deal with the exact representation

of computation and data values. At this level, uninterpreted data flow networks are used in which data are represented by tokens (interpreted data flow networks with exact data values are introduced in further steps of the model refinement). The data dependences are modeled at this level by the stochastic behavior of the network. We restrict our investigation to uninterpreted models.

In the following, first the data flow and Petri net notations are reviewed then the transformation itself is outlined.

2.1. The data flow notation

A data flow network (DFN) consists of nodes and unidirectional channels, where nodes represent computation and data processing, channels represent the data transmission between the nodes and from/to the environment. The possible activities of a node are modeled by firings. If a firing fires, the state of the node changes, some tokens are removed from the input channels of the node and some tokens are inserted into the output channels of the node. A firing is associated with a delay which represents the time needed to complete the activity. During this time, the node is in a working state preventing the start of other firings. The firing delay can be given either by its distribution function or by its exact value (deterministic firing delay). To assure the analytical evaluation of the network, the distribution function of the firing delay is restricted to be negative exponential.

Thus, a node of the DFN is defined by the tuple of

- set of input channels;
- set of output channels;
- set of states with a distinguished *working state*;
- set of firings where each firing is described as follows: the state and the number of tokens in the input channels required by the firing, the state reached after the firing, the number of tokens inserted by the firing into the output channels, and finally the priority and the time function of the firing delay.

The DFN is a composition of the nodes by connecting the channels to each other (each channel connects exactly two nodes, self loops are not excluded). It can be given by the

- set of nodes;
- set of channels;
- initial state of the network (composed of the initial states of the nodes and the number of initial tokens in the channels).

The interaction between the environment and the DFN is taken into account either by modeling the environment (closed network) or by representing the events of the environment by dummy input (output) nodes which have only output (input) channels connected to the channels of the DFN and firings inserting (removing, respectively) tokens into these channels.

A firing of a node is enabled if the node is in the state required by the firing and its input channels contain the necessary number of tokens (i.e. all data required for the activity is present). If an enabled firing fires, first the given number of tokens is removed from the input channels and the state of the node changes to the working state (*start event* of firing). After the firing delay, tokens are placed into the output channels and the state of the node changes to the one given by the firing (*end event*). The selection between the enabled firings of a node is based on the priorities (selection between the firings at the same priority level is random).

In the network, the firings of different nodes can fire parallel, the order of start events is random (since each channel connects exactly two nodes, the nodes are not in conflict with each other). The actual state of the DFN is given by the states of the nodes and the number of tokens in the channels.

To avoid to deal with all changes in the state of the network we are interested only in the states which exist for nonzero time (tangible states). Accordingly, the *computation* (operation) of the network is defined as the series of tangible states of the network together with the time values when the state changed. (Note that tangible states of the network are the ones when start events of enabled firings were executed and the nodes are either in working state or in idle state waiting for tokens in the input channels. If all nodes are in idle state then a deadlock is detected.) The end events of a firing and the successive start events of (other) firings triggered by this end event are grouped into a compound event which changes the tangible state of the net.

2.2. The device handler example

Our example is a device handler in a computer system which controls the data access of a disk subsystem. It consists of a disk, a cache unit, a cache controller, the handler software and two user processes (the environment). The data flow network is presented in Figure 1. The boxes represent the nodes, the arcs represent the unidirectional channels between the nodes. The net can be composed easily using a data flow editor [2].

The application processes can initiate read and write requests independently. The device handler software (split into two parts, handlers of the read and write requests) collects the requests and selects the one being served. At a given time, one application task can be served. The requests are forwarded to the cache controller which selects an activity again and sends the corresponding signal to the cache. The cache organizes the data transfer. If necessary, the cache initiates a read or write operation of the disk. In the example, the disk always performs the requested opera-

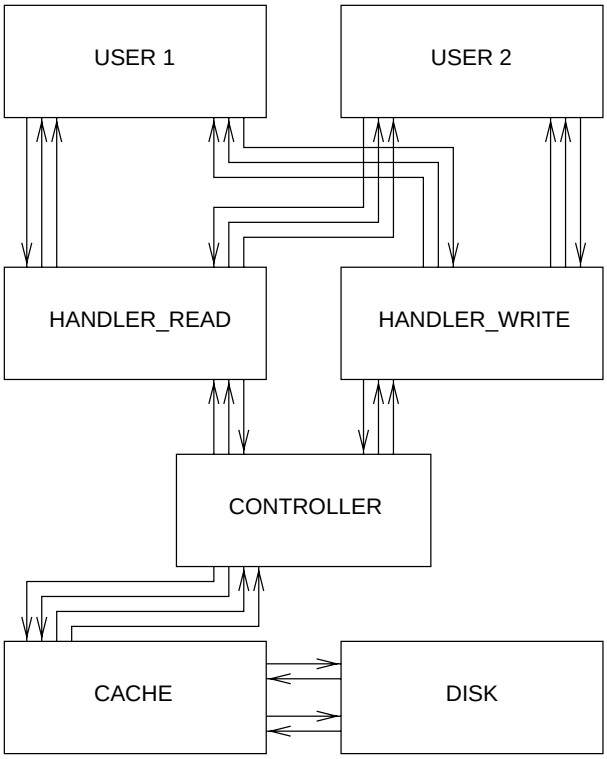

Figure 1. The device handler

tion and sends back the data or an acknowledge signal.

Both write-through and copy-back cache strategies were implemented (in two models), in order to compare them. The write-through cache works in the following way. Write requests refresh the content of the cache but are also forwarded to the disk. Read requests are served either using the content of the cache (cache hit) or first accessing the disk and then updating the cache. The operation of the copy-back cache is more difficult. If the requested data is in the cache (cache hit) then the request can be served without accessing the disk. If the data is not in the cache, first it has to be examined whether the cache contains only dirty data (data which has a valid copy only in the cache). If the cache is full of dirty data, first some part of it has to be written to the disk to free some space for the new data. Consequently, if the cache is full, write requests are served including a disk write then a cache write operations, otherwise only a cache write has to be performed. If the cache is not full, read requests mean a disk read then a cache write operation (enter the new data into the cache), if the cache is full, an additional phase is required to write the disk (free some space, Figure 2).

If the handler returns an error message indicating that the requested operation cannot be performed (since the device is busy due to the request of the other user process) then it has to be retried later. Similarly, if the controller returns an error signal then the handler will retry the request.

Each node is described in an uninterpreted data flow language. It defines the name of the node, the input channels, the output channels, the set of states and the set of firings. As an example, let consider the simple node representing the operation of the disk:

```
NDISK
Xi [read_disk, write_disk]
Xo [read_ok, write_ok]
St [idle]
r1 idle,[1,0],idle,[1,0],e1,1
r2 idle,[0,1],idle,[0,1],e2,1
EDISK
```

The node name is DISK, it has two input channels ($read\_disk$ and $write\_disk$) and two output channels ($read\_ok$ and $write\_ok$). The number of states is one. The firings are given as 6-tuples of the state and the vector of tokens in the input channels required by the firing, the state reached after the firing and the vector of tokens put into the output channels of a node, and at the end the time and priority parameters. E.g. firing $r1$ changes the state $idle$ of the node to $idle$ again while removing 1 token from the channel $read\_disk$ and inserting 1 token into the channel $read\_ok$ (modeling that the disk served the request in a given time then returned to its idle state). Its time parameter is $e1$ (parameter of the negative exponential distribution function), its priority is 1.

The states and firings can be graphically represented in a form similar to the one of state machines. In Figure 2 the firings of the simplified CACHE node are presented (copy-back, read access).

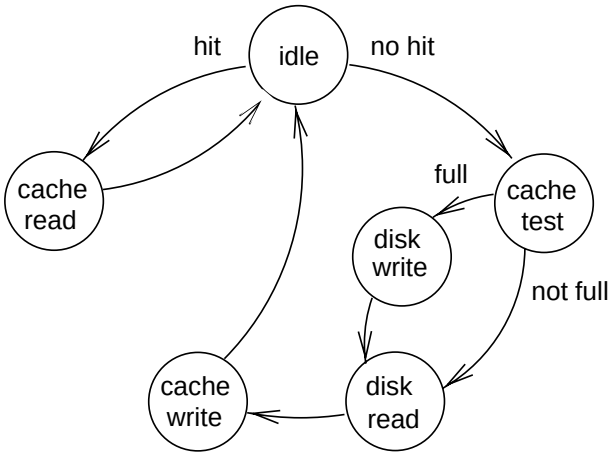

Figure 2. Read operations of the copy-back cache

2.3. The Petri net notation

Our goal is to analyze the data flow networks indirectly, by transforming them into a proper type of Petri nets. Since our DFN is timed, with deterministic constant or negative exponentially distributed firing delays, a class of timed stochastic Petri nets, the Deterministic and Stochastic Petri Nets (DSPN, [8]) was selected.

A Deterministic and Stochastic Petri Net (as an extension of regular Petri nets [9]) is composed of places, transitions and arcs directed from a place to a transition or from a transition to a place. Places connected to a given transition form its input places, places connected from the transition are its output places. The state of the net (called a marking) is given by the number of tokens in the places. Activities of the net are represented by firings of transitions removing tokens from their input places and adding tokens to their output places, the number of transferred tokens is given by the weight of the corresponding arcs. Conditions of firings are represented by the number of tokens in the input places: a transition is enabled if its input places contain the necessary number of tokens (to be removed by the transition).

The DSPN is a tuple of

- set of places
- set of transitions
- set of directed arcs (flow relation) between places and transitions including inhibitor arcs (not used in our model)
- weight function of the arcs
- priority and time function of the transitions
- initial marking of the net.

In DSPN, transitions are immediate or timed ones. Timed transitions are associated with deterministic or exponentially distributed firing delays. Timed transitions have zero (lowest) priority while immediate transitions have higher priorities. An immediate transition fires if it is at the highest priority level among the ones being enabled (if there are more, one of them is selected using random switches assigned to the transitions), a timed transition fires if it is continuously enabled during its firing delay sampled according to the given distribution.

The firing policy is *race with enabling memory*. Each transition is associated with a timer. If the transition becomes enabled then it samples a firing delay using its distribution function and sets the timer to the sampled value. When the transition is enabled, the timer counts down, if it reaches zero then the transition can fire. If the transition is disabled (by the firing of a conflicting transition) then the timer is reset (a new delay has to be sampled when it becomes enabled again).

To find the correspondence between the DFN and DSPN models, the behavior of the DSPN is represented in a similar way like the behavior of the DFN. We define the *computation* of the network as the series of tangible markings of the network together with the time values when the marking changed. In this case, the computation is a modified form of the firing sequence of the DSPN hiding the firings of immediate transitions (which were executed at the same system time). Namely, the firing of a timed transitions and the firings of immediate transitions enabled by this firing of the timed transition are grouped into a compound event which results in a new tangible marking of the net.

2.4. The transformation

Our model transformation maps a DFN to a DSPN. The nodes, channels and firings of the DFN are mapped to subnets of the DSPN, the initial state of the DFN is mapped to the initial marking of the corresponding DSPN.
The transformation is detailed as follows:

Channels: Each channel of the DFN is mapped to a single place. The number of tokens in the channel is represented by the number of tokens in the corresponding place.

Nodes: Each node of the DFN is mapped to a subnet of the DSPN. The non-working states of the node are mapped to unique places. If the node is in a given state then the corresponding place will contain a token. The working state of the DFN node is mapped to a set of places such that each firing of the node is associated with a place. If the node is in the working state after the execution of the start event of a firing then the corresponding place will contain a token.

The start events of firings are associated with immediate transitions with priority inherited from the firing (the value of the random switch is fixed). The end events of firings are mapped to timed transitions with zero priority; the time parameter is inherited from the firing.

The flow relation of the resulting subnet is defined as follows. The input places of an immediate transition corresponding to the start event of a firing are the places representing the input channels and the non-working state of the node required by the firing. The output place is one of the places corresponding to the working state of the node. This place is also the input place of the timed transition corresponding to the end event of the firing. The output places of this timed transition are the ones representing the output channels and the non-working state (result of the firing). The weights of the arcs connected to/from the places representing the channels are given by the number of tokens transferred by the corresponding firing of the

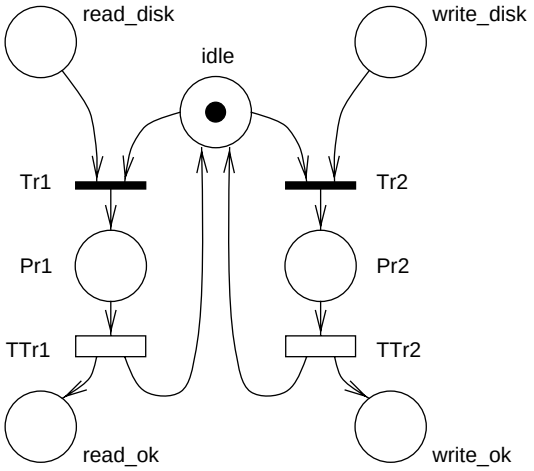

Figure 3. DSPN representation of the DISK node

data flow node, the arcs connected to the places representing the states of the node have the weight 1.

Initial state: The initial marking of the DSPN is defined using the initial state of the DFN. The places corresponding to the initial states of the nodes contain a token, the places corresponding to the channels contain the same number of tokens as the channels.

As an example, let consider the data flow node of the disk (described in the previous subsection). The transformation results in a Petri net shown in Figure 3. The nonworking state is represented by the place $idle$, the channels were transformed to the places with the corresponding name. The firing $r1$ ($r2$) is transformed to the subnet consisting of the transitions $Tr1$, $TTr1$ ($Tr2$, $TTr2$) and place $Pr1$ ($Pr2$, respectively). The initial state of the node is $idle$, thus the corresponding place $idle$ contains a token in the initial marking.

The DFN to DSPN transformation is performed automatically by a tool *df2pn* [1] which was developed at the University of Pisa. It reads the data flow model given in the uninterpreted data flow language presented in Subsection 2.2. After some checking, first a transformation to an internal Petri net description is performed. The internal description can be transformed by individual packages to the representations used by different tools. The *GreatSPN v1.6* tool [4] was first selected because it provided not only sophisticated analysis but also a graphical representation of the resulted DSPN.

Our model (which is right compact and easy to survey as a data flow network) resulted in a DSPN model consisting of 93 places and 88 transitions, which seems to be more difficult to manage even in this little example.

2.5. Equivalence of the two representations

We want to derive selected properties of the data flow model by analyzing the DSPN representation given by the transformation. These properties are related to reachability analysis like

- existence of dead-lock in the system,
- reachability of selected (tangible) states,
- existence of firing sequences corresponding to selected behavior of the system,

or to timing analysis like

- average or maximum cycle times (if a steady state exists)
- average, maximum, minimum times between selected activities of the system.

To derive these properties, it has to be shown that the behavior of the DSPN and DFN models are equivalent (in the sense that the mentioned properties can be derived). The transformation results in a DSPN which can be characterized by its possible computations starting from the initial marking. These computations can be mapped (using the inverse of the transformation [7]) to supposed computations of the DFN (tangible markings are mapped to tangible states, firings of transitions are mapped to start and end events of firings).

In [7] it was shown that – given the semantics of the model presented in Section 2.1 – the DSPN model is equivalent of the DFN model in the following sense:

A tangible marking is reachable from the initial marking by a computation in the DSPN if the corresponding state (given by the mapping) is reachable from the initial state by the corresponding computation in the DFN.
Conversely, a tangible state is reachable from the initial state by a computation in the DFN if there is a marking reachable from the initial marking by a computation in the DSPN where the marking is mapped to the state and the computation of the DSPN is mapped to the computation of the DFN.

This way, reachability problems related to tangible states of the DFN can be solved using the DSPN model, since the tangible reachability graph (TRG) of the DFN can be derived using the TRG of the DSPN. Timing properties can also be derived (note that the computations include time labels). Additionally, the stochastic behavior of the two models are equivalent in the sense that in a tangible state/marking, the probabilities of the possible successor states/markings are the same in the two models.

The transformation and the above introduced relation are not restricted to data flow networks in which the firing delays are either deterministic or exponentially distributed. However, analysis tools are well elaborated only for this

type of networks. In the general case, evaluation of the network is supported only by simulation.

3. Reachability analysis of the model

To perform reachability analysis, the tangible reachability graph (TRG) of the DSPN model was first constructed by the GreatSPN tool (Table 1). It was mapped to the TRG of the DFN utilizing the following properties:

- Each non-working state of a node in the DFN is represented by a single place in the DSPN; if it is marked then the node is in the corresponding state.
- The working state of a node is represented by a set of places in the DSPN (corresponding to each firing of the node); if one of them is marked then the node is in working state. (It was shown in [7] that the transformation results in a network in which at a given time only one of the places corresponding to the working and non-working states of a node is marked, exactly by 1 token.)
- Each channel is represented by a single place in the DSPN; if it is marked with n tokens then the corresponding channel contains n tokens in the DFN.

Cache strategy	Tangible states	Vanishing states
Write-through	436	41
Copy-back	590	44

Table 1. Size of the reachability graph

Analysis of the TRG resulted in the proof of the following important properties of the system:

- There is no deadlock in the system (the competition of the users for the device is resolved correctly).
- The network is live, there are no dead firings (i.e. there are no unnecessary operations defined in the network).
- The initial state is a home state, reachable from all states of the network (there is no such situation that one of the users, handlers or devices is stuck-at in a given state for ever).
- The network is structurally bounded (the handshake between the users, handlers and devices is correct, there is no overflow of data or requests).

In the TRG, existence (or absence) of given states (e.g. crash) or situations (e.g. loss of messages) can be checked. For example, in our network we checked that there is no such situation in which the user is in the state waiting for the acknowledge of its write request and the handler is still busy serving the read request of that user (the user was provided earlier by an illegal message terminating its read request). To do this, the TRG of the network was examined looking for a state in which the node USER1 is in the state $write$ and the node HANDLER_READ is the state serving request of USER1 ($handle1$). The search was performed using the textual representation of the TRG.

4. Timing analysis of the model

In our measurements, the timing parameters of the network were set to express the differences between the speed of the control, cache and disk operations. The control operations (performed by the handler and cache controller) were associated with unit time, the cache read/write operations were defined to be 10 times slower. The disk operations are the most time consuming (100 times slower). The firing delays are exponentially distributed with the above given parameters. Our goal was not to measure the exact times of the operations rather to compare the advantages of different cache controlling strategies in given workloads.

To get the desired results, transient or steady-state analysis of the network can be performed. Steady-state analysis provides the values of the average throughput of transitions and the distribution of tokens in the places. Based on the set of reachable states (given by the TRG), starting from a reachable state transient analysis of the net can also be done.

The relation of the two networks (Section 2.5) enables to derive some measures of the DFN using the results obtained by the analysis of the DSPN, in the following way:

- Each firing of the DFN is represented by two successive transitions in the DFN (an immediate one representing the start event and a timed one representing the end event of the firing). The throughput of these transitions is exactly the throughput (i.e. the number of executions in unit time) of the corresponding firing of the DFN.
- Each non-working state of a node is represented by a single place in the DSPN. The probability that the place is marked is exactly the probability that the node is in the corresponding state. The probability of the working state can be computed by summarizing the probabilities that one of the places corresponding to the working state of the node is marked.
- Channels of the DFN are represented by single places in the DSPN. The distribution of tokens in these places is exactly the distribution of tokens in the corresponding channel of the DFN.

We derived various properties of the network performing the steady-state analysis of the DSPN. The results were available in the GreatSPN environment (an integrated design environment, in which the results of the indirect analysis are propagated back to the original data flow editor, is

under construction). The results were combined to gain the proper measures of the system. Average execution times of the operations, average access time of the device were computed using different cache strategies and workloads.

4.1. Single user access

First the average times required to perform the disk read and write operations were measured. The model of the user process was changed to perform the given operation continuously, this way using the throughput of the corresponding firing the average access time could be directly derived (1/throughput). The 9 different situations were set by changing the priorities of the firings in the model of the cache. The results are presented in Table 2.

Operation	Cache parameters	Av. time
Write	Write-through	118
	Copy-back, hit	17
	Copy-back, no hit, full	119
	Copy-back, no hit, no full	18
Read	Write-through, hit	17
	Write-through, no hit	118
	Copy-back, hit	17
	Copy-back, no hit, full	220
	Copy-back, no hit, no full	119

Table 2. Average access times

Note that the most time consuming operation is the reading of the copy-back cache when it is full. The disk has to be accessed two times (to free some space in the cache for the new data, and read the new data itself). Writing a full copy-back cache with no hit requires the same operations as writing the write-through cache (the examination of the full state of the copy-back cache needs an additional cycle which explains the small difference).

The average access times depend on the full and hit ratios of the cache. The *hit ratio* is defined as the probability that the data is found in the cache. The *full ratio* is the probability that the copy-back cache contains only dirty data (data which have a valid copy only in the cache), i.e. before entering new data to the cache (by a write or read request) some part of it has to be written to the disk. (Of course, in given applications these parameters are usually not independent. Both depend on the locality of the data accesses and on the size of the cache.)

To highlight the dependences, the average write access time of the user was measured setting different cache parameters (the switching probabilities of the firings corresponding to the selection of the cache operations were changed accordingly). The results are depicted in Figure 4.

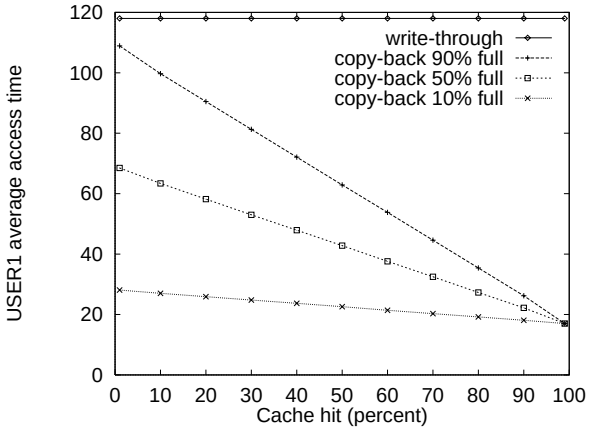

Figure 4. Write access by a single user

Note that the measures of the write-through cache are worse than that of the copy-back cache. If the hit ratio of the copy-back cache increases, the dependence of the access time on the full ratio of the cache decreases (a time-consuming disk access is only needed if there is no hit and the cache is full).

4.2. Results of different workloads

To examine the effects of the workload, three different scenarios were measured.

In the first scenario, both users perform write accesses. USER1 writes the device continuously, while USER2 performs a write access and then spends time with computation (working time). It was investigated how the average access time of USER1 depends on the average working time of USER2. Write-through cache and copy-back cache with 50% hit and 50% full ratios were investigated (Figure 5).

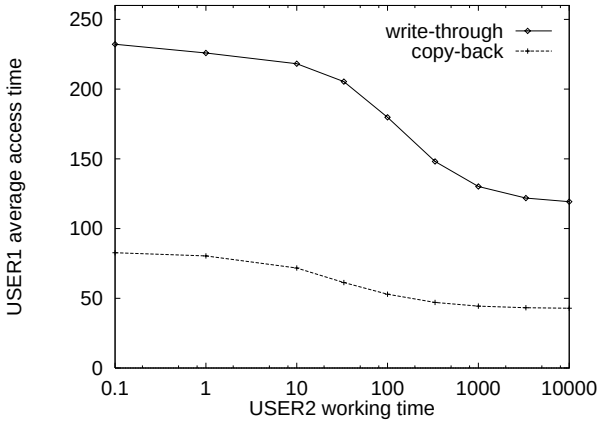

Figure 5. Write access of a user depending on the working time of the other user

If the average working time of USER2 is low then the average access time of USER1 is approx. double of the access time corresponding to the scenario when it is the single user accessing the device (i.e. the handler selects one of the users randomly). If the working time is extremely high, the single-user access time is approximated.

The next two scenarios present two workload situations. In the first (Figure 6) USER1 writes the device continuously while USER2 performs read accesses. In the second case (Figure 7) USER1 reads the device, USER2 reads or writes (selecting randomly). In these scenarios, the performance of the write-through cache is sometimes better than that of the copy-back cache, especially if the hit ratio of the copy-back cache is small, the full ratio is high and the cache is read in the majority of the accesses (second scenario).

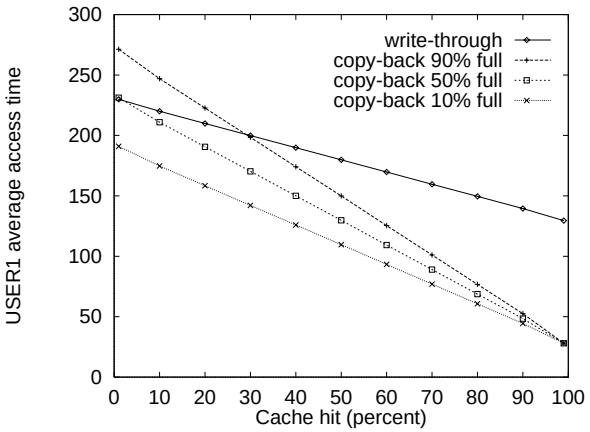

Figure 6. USER1 writes, USER2 reads

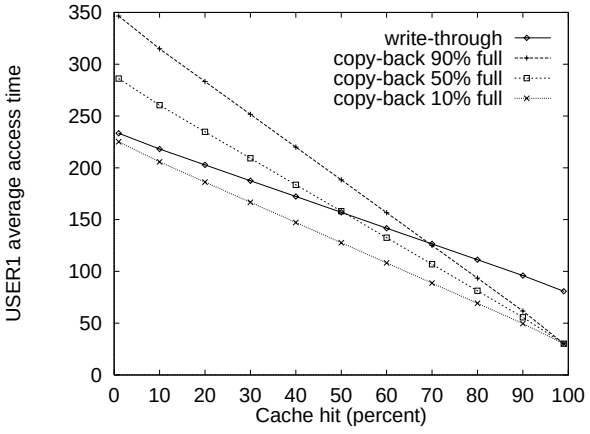

Figure 7. USER1 reads, USER2 reads/writes

5. Conclusions

Data flow networks have advantageous properties like modularity, compactness, support of direct implementation, which make them attractive to system designers in the early design phases. We showed by an example that the lack of widely used analysis tools can be avoided by an automatic model transformation and indirect model evaluation. The equivalent Petri net preserves the timing and reachability properties of the model (which are of utmost interest) and ensures the validation supported by sophisticated analysis packages. Performance measures as average access time, execution time and safety measures as liveness, existence of deadlocks, unreachable states can be derived.

We restricted our investigations to the early phases of system development when the application of uninterpreted data flow networks provides satisfactory results for the system designer. In further phases, by stepwise refinement, interpreted networks can be introduced in which data dependences can be modeled. The elaboration of a similar transformation and indirect model analysis for interpreted networks (e.g. by using colored Petri nets, process algebras) is one of the tasks of future research and development.

References

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

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

[3] A. Bondavalli, L. Strigini, and L. Simoncini. Dataflow-like languages for real-time systems: Issues of computational model notations. In *Proceedings of SRDS-11, Houston, Texas*, 1992.

[4] G. Chiola. A graphical net tool for performance analysis. In *Proceedings of the 3rd International Workshop on Modeling Techniques and Performance Evaluation, March AFCET, Paris, France*, 1987.

[5] G. Csertán, C. Bernardeschi, A. Bondavalli, and L. Simoncini. Analysis of temporal properties of data flow networks. In *Proceedings of the 12th IFAC Workshop DCCS95, Toledo, Spain*, pages 153–158. Elsevier Science Ltd., 1994.

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

[7] I. Majzik. On semantics and temporal analysis of data flow networks. Internal report IR-1/94-L.S./I.M., University of Pisa, July 1994.

[8] M. A. Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In G. Rozenberg, editor, *Advances in Petri nets*, volume 226 of *Lecture Notes in Computer Science*, pages 132–145. Springer Verlag, 1987.

[9] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.