INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Performance Oriented Specification for Heterogenous Parallel Systems using Graphical Based Specifications

Herwig Unger*, Bernd Daene†

TR-TR-95-043

08/19/1995

*University of Rostock, Germany; hunger@informatik.uni-rostock.de

†Technical University of Ilmenau, Germany; bdaene@theoinf.tu-ilmenau.de

Abstract

Today, multiprocessor systems can be used even for the solution of small problems. In contrast to this advantage in the development of hardware solutions there are only a few methods to specify and to generate efficient parallel programs especially in the area of heterogenous systems.

In the report we intend to show that Petri Nets are a suitable description language for doing so. An important point in this favour is that Petri Nets can represent both aspects influencing the quality of a solution in an uniform model: the software and the hardware on which the generated program will be executed. In that way the executable program can be derived by compiling the corresponding part of the model. Therefore powerful transformations of a given Petri Net are required in an iteration process. Thats why a classification about such transformations is given in the main part of our contribution, furthermore an new one will be introduced. Because run time input data strongly influence the performance a possibility of a dynamic implementation arising from such a transformation will be discussed too. Finally we demonstrate and compare the achieved results for one typical example.

1 Introduction

There are several approaches in order to improve the performace of a given problem solution. On one side a better time efficiency yields from an improvement of the hardware, for instance the use of more memory or coprocessor units. On the other side more important results can be achieved by using parallel computer architectures. Today different types of parallel architectures are available even for smaller problems. In a first approximation all parallel systems can be considered as a number of processing units, a communication network and several software units. The effective use of these structures requires a lot of new approaches and methods in order to solve the following problems and optimizing the results:

- identify parallel executable parts
- map these tasks to the processing units
- define the communication structure and
- determine the communication

Difficulties in the software development process described above often arise from the following facts:

- One cannot think parallel.
- There are only a few standardisation projects for parallel HLL
- Portability of programs can not be achieved because of the different hardware, especially in the communication structures.
- The complexity of problems is often very high (class NP)

To avoid these disadvantages in the development process of parallel software several methods were derived increasing the acceptance of parallel processing machines. They are based on the use one or more of the following four main ways:

- parallelizing existing serial standard software
- specification within a parallel HLL
- specification with an objectoriented, parallel HLL
- modelling with a graphical description language such as graphs, Petri Nets, dataflow diagrams and compiling these solutions into executable code

A lot of parallel programs can be generated for solving the same problem even from the same algorithm, but it is very difficult to compare the different results. In most cases this comparision can be made only by time and cost extensive run time tests. So the use of other new approaches is indicated.

2 Petri Net based Modelling and Specification

There exist a lot of possible quality arguments and needs showing once more the complexity and needs for a good development of parallel software:

- the execution of the program in real time (mostly)
- the execution of parts in a special time interval
- the execution of special parts on a special group of machines
- low communication (especially in workstation clusters, where we have a very slow and therefore very expensive communication, which often does not meet the requirements of the user)
- the waiting times of all processes should be minimized
- the number of processes should be equal to the number of really parallel executable parts (fireable transitions) of the solution

It seems from the high number of difficulties and new features that common programming languages can not meet the requirements for the effective development of parallel programs. In such a way graphical descriptions became more and more important. Finally, from the topics shown above it should be evident that effective parallel programming means to optimize both hard- and software structures. In such a way a model is needed which can represent these aspects in **one** description. Petri Nets have proved to be an efficient tool to represent complicated systems, especially because time dependencies and stochastics are included (see [4]). Therefore the authors suggest a way basing on a (at least) three level Petri Net modell of the system architecture derived from the results described in [5]. Another advantage should be that during the whole implementation all other problems can be solved in only one software unit based on the language of nets.
The lowest layer of such a model will be built by the physical structure of the communication topology with its input and output points from i/o devices of the machines and an interface, if there is a configuration possibility with switches (for instance δ-networks). The next layer represents all functions which are hidden in the operating system, especially the routing of information between the concurrent working communication channels, waiting queues and scheduling informations. The reader should note, that these informations have to be a part of a hardware library of the used system which can be reused each time. The third part will be built by the application of the user which was former the only part of the model description. The interfaces between the layers are transitions. In such a way the elements of the second layer contain a refinement and an additional connection of those ones from the first level and so on.

As an example we consider the high performance workstation cluster suggested in [9]. In order to improve the performance of the ethernet connection a set of workstations is connected with a very efficient hardware construction (see figure 1, 2). When using a concurrent communication structure the transmission will be about two times faster for the two dimensional structure. This result can be obtained from theoretical considerations,

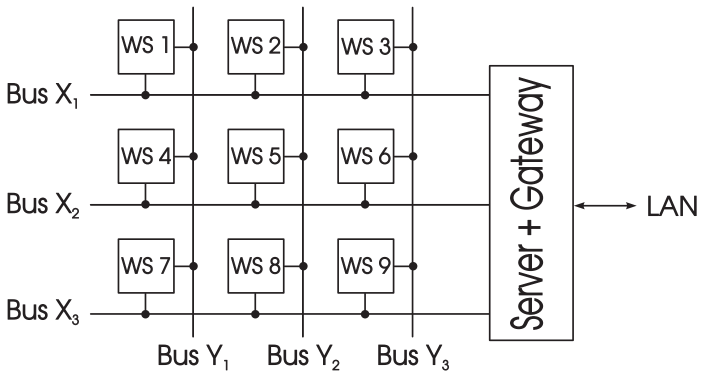

Figure 1: 2-dimensional Workstation Cluster Architecture

practical experience or from a simulation using our proposed method. A very rough picture of only one node is given in figure 3. It shows the two lowest layers of the hardware; informations will be represented by tokens. The color of the token represents the final destination of the packets, furthermore the token can be considered as a container for any appended message. The multiplicity of the arcs leaving the places representing the exclusive usable bus (global capacity 1) describes the routing of the information between the nodes and buses. On the second level, each node contains buffers and queues for the incoming and outgoing messages and some software for selecting, receiving and sending informations from tasks running at this processing unit. The CPU place and the testing arcs describe the timesharing within the node for several tasks. All time dependend parameters must be determined by statistical measurement and are connected with the parameters of the transitions. The transitions t_o and t_i form the interface to the application layer in the described way.

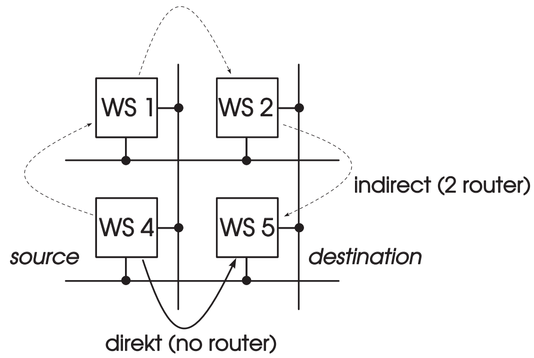

Figure 2: Concurrent Communication in the Cluster

While only considering performance improvements for the architecture at all, a stochastic firing process can model the access to the communication ressources. Even this very easy model is suitable for a more performance oriented description of the system architecture. It can be shown that the amount of packets waiting for transmission in the buffers of a node is strongly related to the performance of the communication network for a given application. By considering the number of tokens in the places of the model corresponding to the buffers it can be proved that the performance of the architecture is really about two times higher than the performance of a one dimensional cluster.

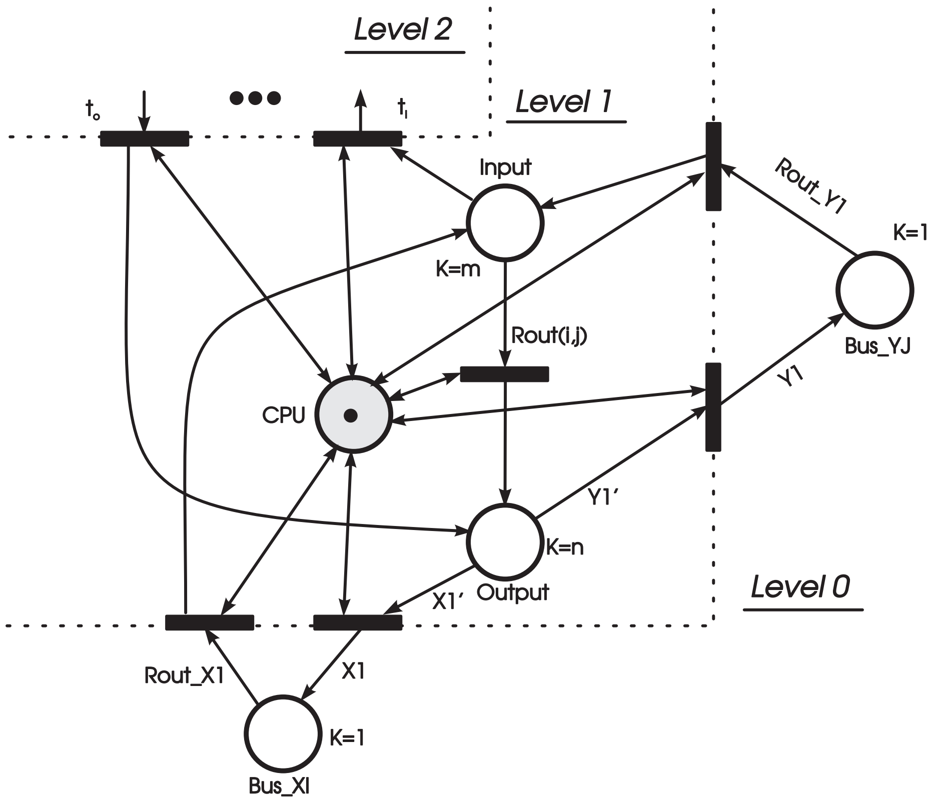

Figure 3: (Colored) Petri Net Model of one Processing Unit

More information about the systems architecture we can get by reachability analysis and other analyze methods. Main properties such as time behaviour and bottlenecks can be predicted with a good quality. If the architecture does not meet the requirements the responsible parameters can be changed and a better solution can be derived in an iterative development process (see figure 4).
Note that the suggested simulations/analyses always need a first mapping of elements or groups of elements to the processor executing them. This is a result from the compilation of the given net described in the following section. Another compilation is needed before each new iteration step which can be influenced by several free selectable options derived from the results of the quality considerations.

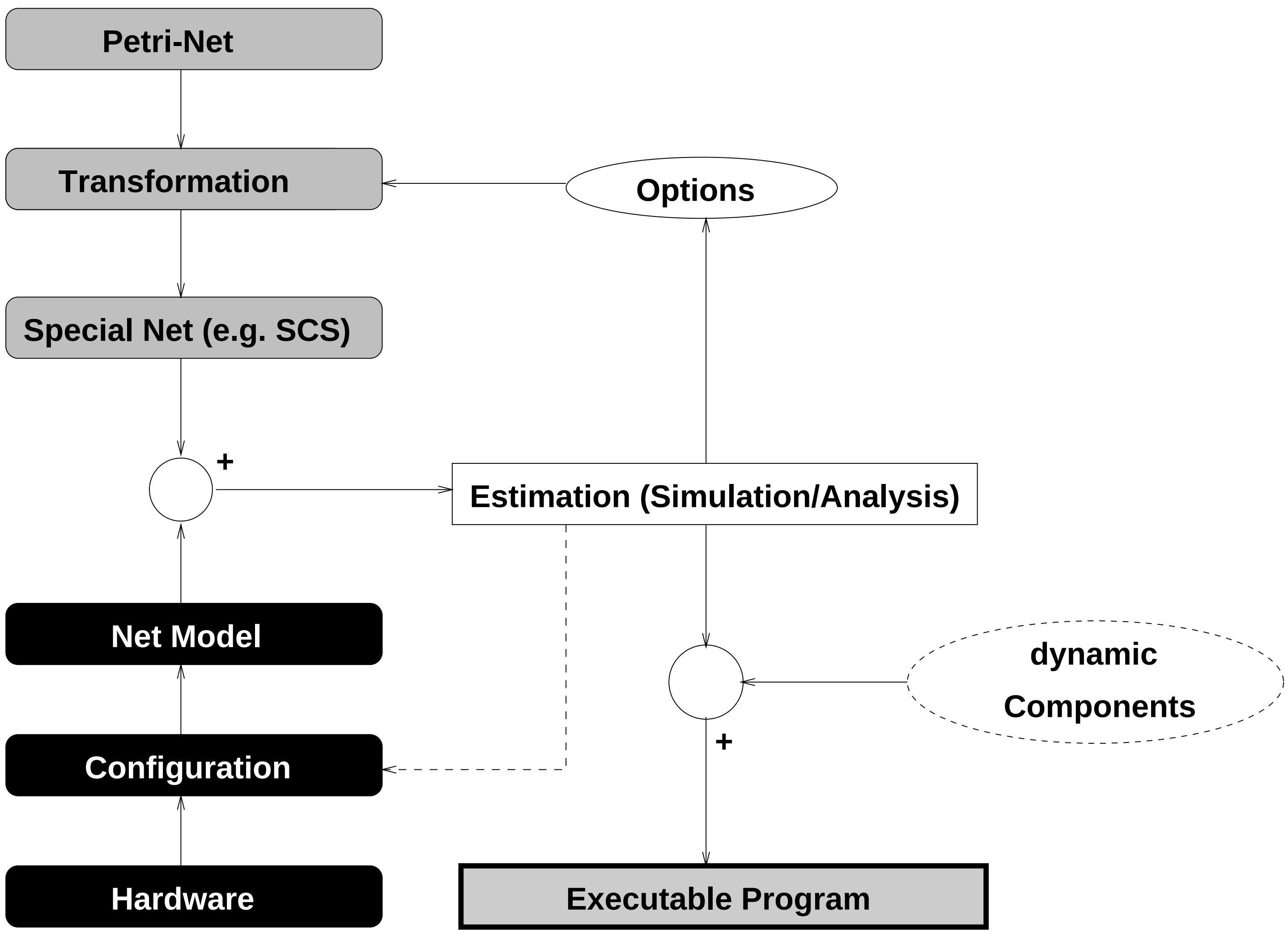

Figure 4: Suggested Specification Approach

3 Approaches for the Compilation of Petri Nets

As shown, Petri Nets can be established to be an effecient tool to represent complicated systems. Nevertheless, in general it is not easy to transform a technical system, given as a Petri Net, for implementation on a multiprocessor system. One purpose of the present contribution is to present a procedure for doing so. Therefore it is useful to find algorithms to solve a lot of problems in a transparent way or hide some parts for a wide group of users. While doing so Petri Nets as one kind of a graphical description language became more and more interesting for modelling parallel software solutions (see [7]) and their acceptance will be increasing.

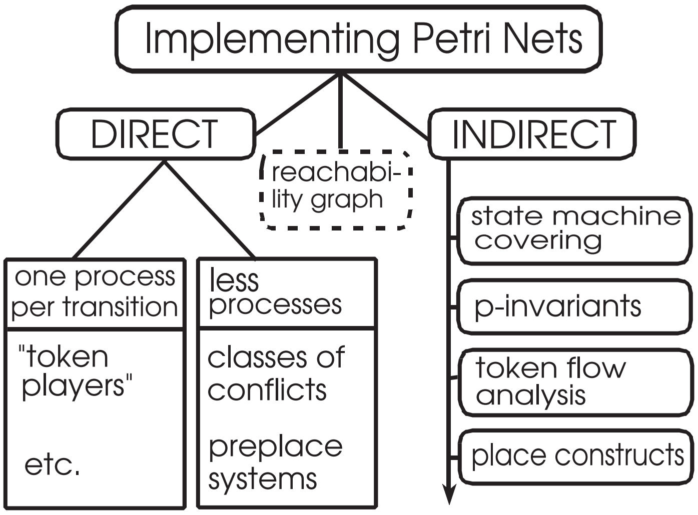

Figure 5: Overview about Exisiting Implementation Approaches

At first the needs of implementing Petri Net solutions will be considered by the following definitions. Implementing a given Petri Net means:

- ... to implement a parallel program with the same behaviour of its input and output data streams like the net. -1-
- ... to find out structures of a parallel program corresponding to structural units of the Petri Net in such a way, that the state (or a class of states) of the program can be derived from the marking (or a class of markings) of the net and vice versa. Doing so each implementation of a given Petri Net is a kind of a fast simulation too. -2-
- ... to write a program, which shall transform the marking m_0 step by step into markings m_i by checking and firing the transitions of the net in accordance to the firing rule. -3-

Each of the given definitions can be used in order to find several approaches implementing the net.
But today there are only a few approaches for implementing Petri Net models on different multiprocessor architectures (a classification is also given in figure 5).

From the authors point of view all known methods for Petri Nets today fall into two basic types. The first one - so called direct type- means to implement one process for every transition (for instance [10] or [14]) is closely related to definition -3-. The second, indirect one is to cover ([7]) or to decompose a given Petri Net by state machines, and then to implement one process for every state machine (see definition -2-). Especially if a Petri Net has a high number of transitions, the first method yields in each case a solution with a plenty of superfluous processes or processing time and a large communication overhead. In general, the second group of methods generate a more efficiant code, but, in contrast to the first one, it does not apply to all Petri Nets.
So in [1] an interesting approach can be found using P-Invariants in order to cover a net with state machines. This approach requires that the multiplicity of all arcs must be one otherwise additional places will be inserted such that the efficiancy of the generated code can be compared with those of the direct methods.
For persistent, bounded Petri Nets another idea can be used. It is justified by the experience that, at least in many Petri Nets arising from practical problems, the number of circulating tokens is much smaller than the number of transitions. Very often these tokens represent physical objects and the movement of these objects activates several procedures. Using a special transformation one state machine for each token circulating in a given net will be generated with a number of additional places, forming a system of state machines which simulates the original net and can be implemented very easy (for details see [12]).
The main disadvantage of known approaches is the transformation of a subset of places into global data objects in a shared memory. These data objects normally contain integer values corresponding to the number of tokens in the places. Accessing the data objects by more than one process causes a lot of management problems and aggravates real parallel work of these processes. In the end a lot of technical systems (like transputer systems or PVM implementations [1], see [13]) require a client server relation instead of a shared memory for solving this problem and so the number of parallel working processes is increased. So another transformation is introduced in this report avoiding these problems, giving good possibilities for a good performance of the derived program and allowing an expanded use of the method from [1]. It is based on the substitution of each place by a special place construct in accordance to the given structure of the net.

[1] **P**arallel **V**irtual **M**achine for UNIX clusters from the Oak Ridge National Laboratory (see: Sunderam, V.S.: PVM.- in: Concurrency: Praktice and Experience, Dec. 1990, pp. 315-339, Atlanta 1990)

4 Transforming Petri Nets using Place Constructs

4.1 Definitions, notations and preliminary considerations

In order to introduce the transformation below only simple P/T-Nets will be used. But the reader should note that the idea can be generalized for other classes, especially colored nets, too.

As usual a Petri Net Φ is a 5-tupel (P, T, F, V, m_0) such that

(i) P, T are disjoint finite nonempty sets, the sets of places and transitions, respectively
(ii) $F \subseteq P \times T \cup T \times P$, the set of arcs
(iii) $V : F \to \mathbf{N}$, the multiplicity function
(iv) $m_0 : P \to \mathbf{N_0}$, the initial marking
($\mathbf{N}$ and $\mathbf{N_0}$ denote the sets of positive and nonnegative integers, respectively.)

For $t \in T$ $(p \in P)$ let

$$Ft = \{p | p \in P, \quad (p,t) \in F\}$$
$$Fp = \{t | p \in P, \quad (t,p) \in F\}$$

and

$$tF = \{p | p \in P, \quad (t,p) \in F\}$$
$$pF = \{t | p \in P, \quad (p,t) \in F\}$$

An unmarked Petri Net is a 4-tupel (P, T, F, V) subject to the conditions (i), (ii), (iii) of the definition of a Petri Net.
We find it convenient to continue V on $F \subseteq P \times T \cup T \times P$ by defining $V(f) = 0$ for $f \in (P \times T \cup T \times P) \setminus F$. Then F is uniquely determined by V, and so a Petri Net can be described by the 4-tupel (P, T, V, m_0), as well.
A transition $t \in T$ is able to fire at a marking m if for every $p \in P$, $(p,t) \in F$

$$m(p) \geq V((p,t))$$

Firing $t \in T$ at m means to substitute m by m_{new} where

$$m_{new}(p) = \begin{cases} m(p) - V((p,t)) & : \quad (p,t) \in F \\ m(p) + V((t,p)) & : \quad (t,p) \in F \\ m(p) & : \quad else \end{cases}$$

for any $p \in P$.

A Petri Net (P, T, F, V, m_0) is said to be a state machine if

(i) $|Ft| = |tF| = 1$ for any $t \in T$
(ii) $V(f) = 1$ for any $f \in F$
(iii) $\sum_{p \in P} m_0(p) = 1$

Obviously, a state machine is 1-bounded and conservative.
A Petri Net (P, T, F, V, m_0) is defined to be a system of concurrent state machines (briefly an SCS) if there is a collection of pairwise disjoint state machines $\Sigma^i = (P^i, T^i, F^i, V^i, m_0^i)$; $i = 1(1)k$ such that

(i) $P \supseteq \cup_{i=1}^{k} P^i$
(ii) $T = \cup_{i=1}^{k} T^i$
(iii) $F^i = (P^i \times T^i \cup T^i \times P^i) \cap F$ for $i = 1(1)k$
(iv) V^i is the restriction of V on F^i, for $i = 1(1)k$
(v) m_0^i is the restriction of m_0 on P^i, for $i = 1(1)k$.

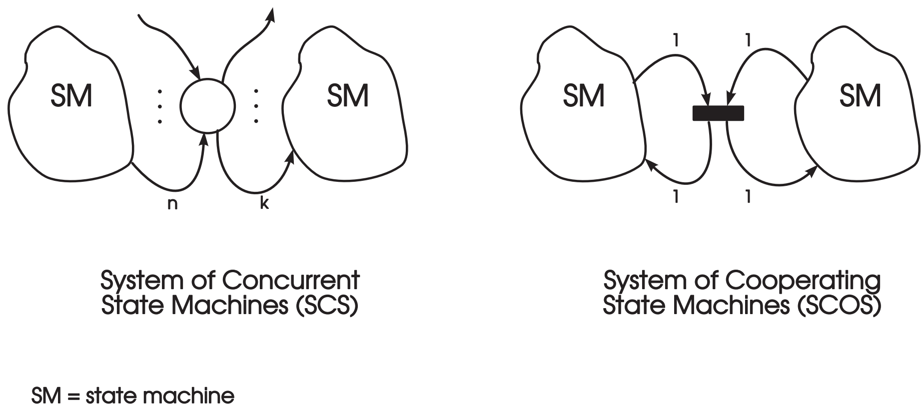

Figure 6: Systems of State Machines

If the state machines are connected by merging two ore more transitions from different state machines, than such a system is called a system of cooperating state machines (see figure 6).

For modelling automation systems it is necessary to add some components to the standard Petri Net definition in order to describe the input and the output of data ([3]):

(1.) $\mathbf{w_x}$,
a set of boolean expressions associated with the transitions.
If $t \in T$, $w_x(t)$ is considered to be an additional condition to fire t.

(2.) $\mathbf{w_y}$,
a set of boolean output variables connected with the places of the Petri Net. $w_y(p) \in w_y$ is $TRUE$, if p is labeled.

(3.) $\mathbf{w_a}$,
a set of procedures associated with the places of P.
Procedures are started when a new token reaches the place.

4.2 The Transformation

In the following, a Petri Net transformation is shown resulting in a net with particular properties. It is based on separating conflict structures followed by a transformation of the remaining net. Afterwards, the net can be implemented in a message based manner.

4.2.1 Conflict situations

Conflicts directly influence the transformation of a Petri Net. Places with more than one posttransition are the reason for conflicts in a Petri Net. Such constructs are called static conflict situations. For the present contribution it is necessary to consider several static conflicts in a given Petri Net Φ in a more detailed way (see figure 7). All the structures consist of a set of transitions A and a set of preplaces S of the transitions of A while there is at least one transition to each other one such that they have a common preplace.

All non-free-choice conflict structures result in problems during the (basic) transformation and have to be cut out in a first step described below.

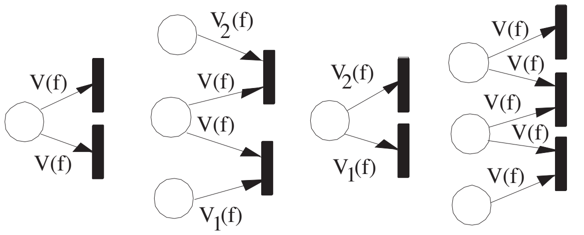

Figure 7: Static Conflict Structures in a Petri Net

Let Π and Θ be set systems for all conflict structures of a given Petri Net with

$$\Pi = \{S_1, S_2, ..., S_h | h \in \mathbf{N}\}$$

and

$$\Theta = \{A_1, A_2, ..., A_h | h \in \mathbf{N}\}$$

The function $K(\Pi, \Theta)$ is defined as follows:

$$K(\Pi,\Theta) = \begin{cases} (\Pi', \Theta') & if \quad \exists i,j : A_i \cap A_j \neq 0 \\ \qquad\qquad with & \Pi' = (\Pi \setminus S_i \setminus S_j) \cup \{S_i \cup S_j\} \\ \qquad\qquad and & \Theta' = (\Theta \setminus A_i \setminus A_j) \cup \{A_i \cup A_j\} \\ (\Pi, \Theta) & if \quad \forall i,j : A_i \cap A_j = 0 \end{cases}$$

Obviously, there is a $k \in \mathbf{N}$ such that $K^k(\Pi, \Theta) = K^{k+1}(\Pi, \Theta)$. In this case $K^k(\Pi_0, \Theta_0)$ is called a maximal conflict set.

For $Q = \{q | q \in P, |pF(q)| > 1\}$, (Π_0, Θ_0) with

$$\Pi_0 = \{M_i | M_i = \{q_i\}, i = 1(1)|Q|\}$$

and

$$\Theta_0 = \{N_i | N_i = \{t | (q_i, t) \in F\}, i = 1(1)|Q|\}$$

is the set of places and their posttransitions which could be the source of a conflict. Furthermore, the connection between some of such sources via their transitions (figure (7d)) is represented in the maximal conflict set $K^k(\Pi, \Theta)$.
In order to get a set with all preplaces of $t \in \Theta$ we modify in $(\Pi, \Theta) = K^k(\Pi_0, \Theta_0)$ the set system Π by $\Pi' = \{p | \exists t \in \Theta : (p, t) \in F\}$.

For further transformation we have to cut out such structures (see figure 8) from a given Petri Net Φ. The main idea consists in a functional separation of the pre- and the postarea of a transition. The fireability of such a transition can completely be tested in the first subnet. The postarea of the transition located in the second subnet only sets tokens on places, if this transition has got a message from the prearea.

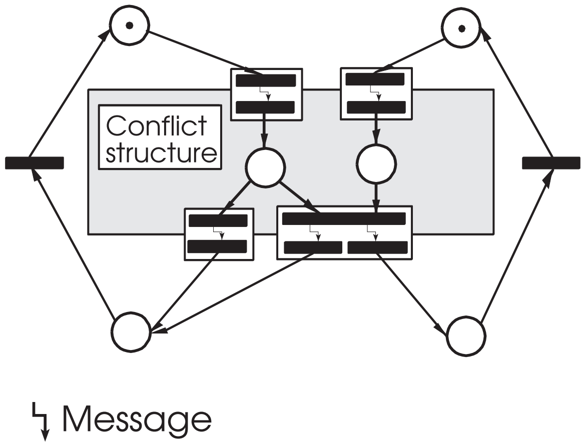

Figure 8: Separation of Conflict Structures

A later discussion shows that only conflicts containing the more difficult subconflict situation from figure 7c must be cut out.

4.2.2 Transformation of the remaining Petri Net

The transformation of the modified Petri Net $\Phi = (P, T, F, V, m_0)$ (a net without static conflict structures) described in this section is carried out in three steps. At first, we define an unmarked place construct $(P'(p), T'(p), F'(p), V'(p))$ for each $p \in P$ of a given Petri Net Φ. After doing so, these constructs will be joined by arcs, and a corresponding marking m' is defined. Thus, we get a corresponding Petri Net $\Phi' = (P', T', F', V', m')$ with $P' = \bigcup P'(p)$, $T' = \bigcup T'(p)$ and $F' \supset \bigcup F'(p)$.

(1.)
Let $p \in P$, $t_{out} \in T$ the only transition with $(p, t_{out}) \in F$ and V_{out} the multiplicity of (p, t_{out}). Then we define

$$u = V_{out} + max(V_i | i = 1(1)|Fp|) - 1.$$

Now each $p \in P$ will be transformed into a place construct with a set of places $P'(p)$ defined by

$$P'(p) = \{p'_0, .., p'_i, x_1, .., x_e | i = 0(1)u, e = 1(1)|t_{out}F|\}$$

For the definition of the sets of transitions and arcs $C_1(p)$, $C_2(p)$ and $C_3(p)$ are defined by:

$$\begin{array}{rcll} C_1(p) & = & \{(a,b,c)| & a = 0(1)V_{out} - 1, b = 1(1)|Fp|, \\ & & & c = a + V_b : a + V_b < V_{out}\} \\ C_2(p) & = & \{(a,b,c)| & a = V_{out}(1)u, b = 0, c = a - V_{out} : \\ & & & a \geq V_{out}\} \\ C_3(p) & = & \{(a,b,c)| & a = 0(1)V_{out} - 1, b = 1(1)|Fp|, \\ & & & c = a + V_b - V_{out} : a + V_b \geq V_{out}\} \end{array}$$

With these definitions let

$$C(p) = \bigcup_{i=1}^{3} C_i(p).$$

Corresponding to the elements of $C(p)$, the following transitions and arcs are added for each $(a, b, c) \in C(p)$ to the sets $T'(p)$ and $F'(p)$, respectively:

$$t_{a,b,c}(p) \in T'(p),$$
$$(p'_a, t_{a,b,c}) \in F'(p) \quad with \quad V((p'_a, t_{a,b,c})) = 1$$

and

$$(t_{a,b,c}, p'_c) \in F'(p) \quad with \quad V((t_{a,b,c}, p'_c)) = 1.$$

Finally, we add for each $(a, b, c) \in C_2 \cup C_3$ arcs with

$$(t_{a,b,c}, x_i) \in F'(p) \quad with \quad ((t_{a,b,c}, x_i)) = 1$$

for all $i = 1(1)|t_{out}F|$.
In a last step places without pretransitions and their postarcs and posttransitions will be removed from the place constructs. An example of such a place construct is given in figure 9.
(2.)
Let $e = 1(1)|t_{out}F|$. Then one $x_e \in P'(p)$ exists corresponding to each of the postplaces $v_1, v_2, .., v_e$ of t_{out}. Furthermore $t_{a,b,c}(v_e) \in T'(v_e)$ are the transitions of the corresponding place constructs. Now we add for all $(a, b, c) \in C_1(v_e) \cup C_3(v_e)$ an arc to F' with

$$(x_e, t_{a,b,c}(v_e)) \in F' \quad with \quad V((x_e, t_{a,b,c}(v_e))) = 1.$$

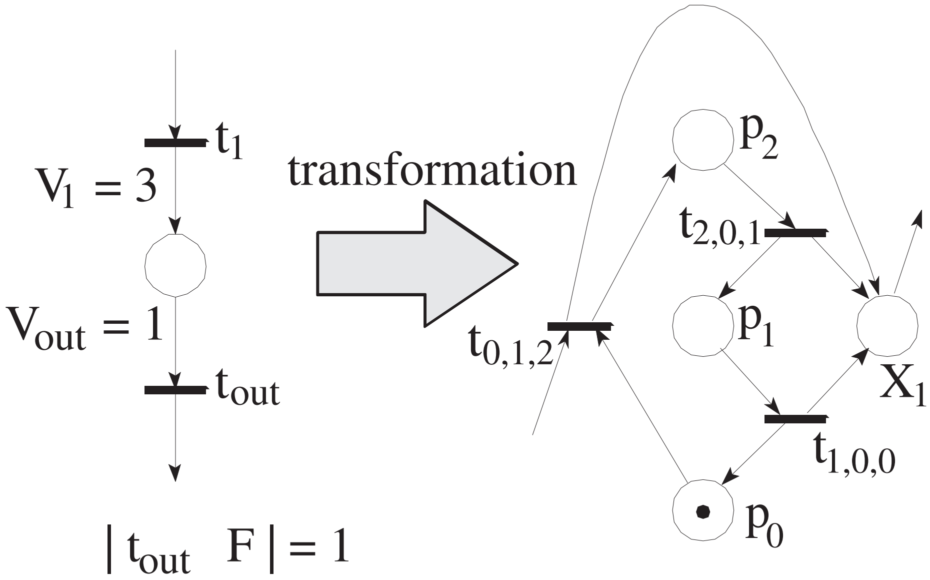

Figure 9: Example of an easy Place Construct

(3.)
A marking m' of Φ' is said to be corresponding [2] to m of Φ if for all place constructs

(i) $\sum_{p'_i \in P'(p)} m'(p'_i) = 1$
(ii) $\forall i, j : m'(x_i(p)) = m'(x_j(p))$
(iii) $\forall i, j : if \quad m'(p'_i) = 1, p'_i \in P'(p)$
$i + m'(x_j(p)) * V_{out} = m(p)$

As a the result of the transformation we get a transformed Petri Net Φ' which simulates the behaviour of Φ. An important property of Φ' is that the multiplicity function equals 1 for all arcs of the net.

[2]there are possible markings among the corresponding markings which do not influence the correct control flow

5 Implementation

5.1 Remarks about Systems of State Machines

The aim of the transformations described above was to decompose a given Petri Net into a special one building a system of concurrent or cooperating state machines (or a mixed structure of both). This aim is also achieved, if the transformed net can be covered with state machines in another easy way.

The reason for doing so is that every state machine can be implemented as a single process. Each of these processes is sequential since any marking of an ordinary state machine contains exactly one tolen and has therefore one fireable transition each time. To implement a transition is a tedious task, and, in general, functions should be used for it. Doing so, the only difference between the implementation of systems of concurrent and cooperating state machines is in the used communication procedure of these functions. Therfore shared data objects or rendesvouz routines are used respectively (see figure 10). These differences and the use of different hardware dependent communication concepts are the reason for achieving advantages using transition functions; all requirements for the communication will be encapsulated and only the transition function has to be adapted to the special hardware and communication structure. Examples for the inner structure of such functions can be seen from [11].

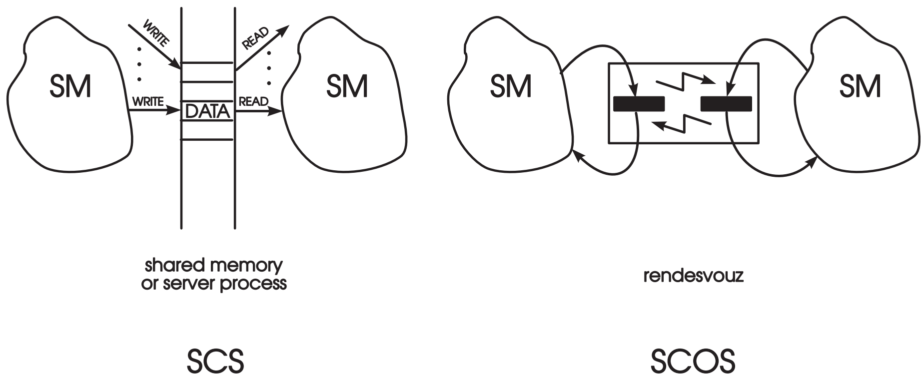

Figure 10: Communication between the State Machines

5.2 Implementing the Transformed Net

The task of implementing the transformed net desribed above comes along with some different requirements than implementing systems of state machines. Implementing Φ' means to find out interpretations for special elements of the given Petri Net. In our case this job falls into two parts: implementing the conflict structures and implementing the transformed remaining Petri Net Φ', which can easily be covered with state machines.

The main problem with implementing conflict structures results from the shared use of a data object representing places with more than one posttransition. Our approach avoids these problems, because all elements of the conflict structure ($K(\Pi, \Theta)$) will be cut out and implemented as a single process, containing all elements for the complete solution of the conflict in a loop. The connection of the conflict structures with the remaining net can be represented by messages, as described above by the shared implementation of a transition in cooperating state machines.

Now let us consider the remaining Petri Net Φ'. One advantage of the described transformation is that the place constructs without the places x_i are state machines. These state machines are connected via x_i and their incident arcs, thus forming so-called systems of concurrent state machines (SCS).
In a first approach these SCS can be implemented by creating a single process of a parallel program for each state machine ([11]). Following this idea, the x_i-elements of Φ' are interpreted as communication structures between these processes.
Places connecting state machines are usually implemented as data objects in a shared memory or a server process ([12]). But resulting from the transformation described above, each x_i has prearcs only relating to transitions in exactly one state machine, and has postarcs only relating to transitions in exactly one other state machine. Therefore, information about the state of any x_i will be managed by only one process and so this communication can be implemented by the use of *send* and *receive* procedures and the belonging message buffers.
A second approach implementing the transformed Petri Net is based on special structure effects of the transformed Petri Net.

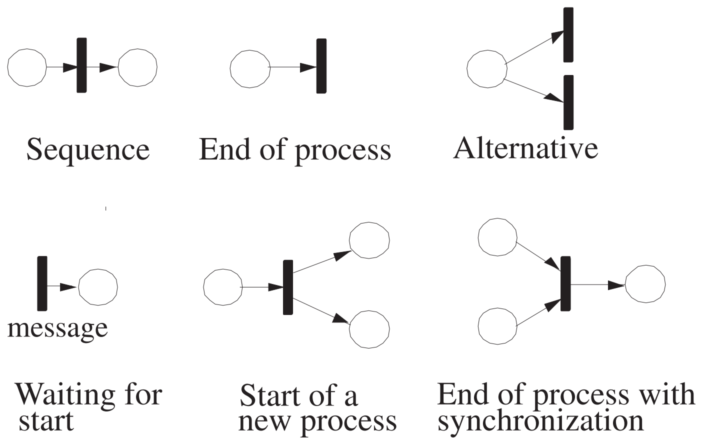

Figure 11: Elements in the Reduced Transformed Petri Net

Let us consider Φ' without the places p_i of P' and without their transitions $t_{a,b,c}$ derived from the elements of $C_2(p)$. It can be shown that such nets consist of six basic elements with an interpretation shown in figure 11. Because the multiplicity of all arcs equals 1, each token in one of the x_i-places corresponds to a set of parallel processes corresponding to the given interpretation of elements. For more than one token we get a superposition of such process groups.
In all cases there is the restriction that in a given moment only one transition of each place construct can fire. This will be achieved by a special interpretation of the p_i-elements of the transformed Petri Net. The marking of these places can be considered as a special value of

a marking of p in the original net. Thus the values can select the fireable transitions and in this way solve the conflicts in the processes. In the parallel program the value of a counter will be implemented by messages circulating between the processes. Only one process can receive the message, and therefore only one process can do the next step corresponding to the firing process of exactly one transition. Leaving the sector of the given place construct the process sends a message with the new counter information and any process that needs this information can receive it.

At last, we have to consider the interpretation of the transitions $t_{a,b,c}$ derived from the elements of $C_2(p)$. Firing one of these transitions entails creating tokens on x_i and processes, respectively. The firing process of these transitions directly depends on firing $t_{a,b,c}$, if $t_{a,b,c}$ derives from C_3 and $c \geq V_{out}$. This algorithm is implemented by creating a new process which receives as its argument the data from the circulating message. The mentioned process creates other new processes, changes the information of the message ($-V_{out}$ for each new process up to the moment when the data are less V_{out}) and sends the updated message to any process requiring it.

The choice of the implementation method depends on the properties of the given net. If the number of places is not too high, the first approach is more effective, a lower number of tokens favours the second method but a mixed use of both methods is possible too.

The reader should note that we have only explained the basic idea and various improvements can still be made. For example, one can involve more subtile HLL structures as if-then-else- constructions or loops ([6]). The latter also permits the possibility to generate a program to control some technical systems represented by a Petri Net involving w_a, w_y and w_x functions ([3]) automatically. Finally note that the described method was explained for Place-Transition nets only but can be expanded for colored nets too.

6 Performance Considerations and Load Balancing

6.1 Performance

For our performance considerations we have used a workstation cluster as explained in the first part of the report. For this architecture a small but typical example is used which contains 18 transitions. A maximum of four of them can be executed in parallel. The size of the procedures for the following measures meets the requirements for a sucessful use of parallel implementation of the Petri Net which can be derived from figure 12. Therefore this figure shows the run time for 1000 cycles of the example. Because the process structure depends on the path used for tracing the net two possibilities were considered in addition to the serial and common parallel implementation (one process for each transition): in the first one the shortest way from a labeled place to the same place back or to dead end was selected building the frame of the process and in the second one longest pathes were used.

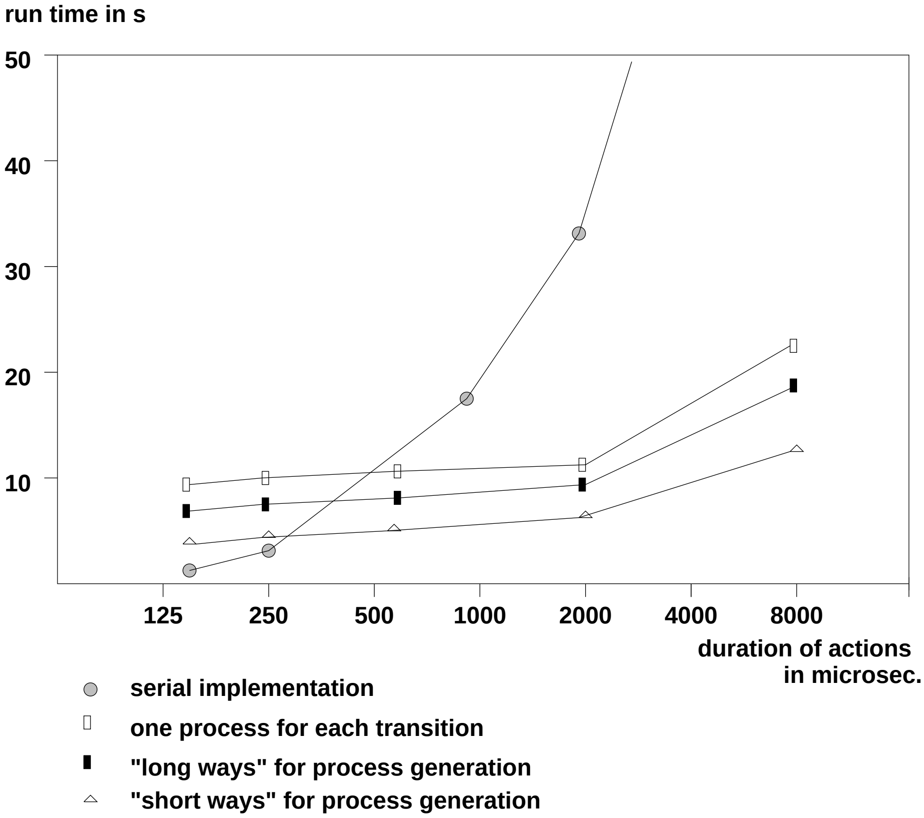

Figure 12: Requirements to the Level of Abstraction of the Net

Because the speedup of a parallel program is the most important factor, in figure 13 an overview for several implementations on a cluster while using 1 to 9 machines is given. Caused by the slowly working communication network, there is no speedup to be seen for the implementation results from making processes corresponding to each (group of) transition(s). In this case the high number of processes yields a high communication expenditure which cannot be served by the network. This is the reason for achieving better results using other approaches. So the method described above, using place constructs generates in both cases only 6 processes which can work more effectively. Furthermore it can be seen that improvements from an optimization of processes within this method only make small changes in the results.

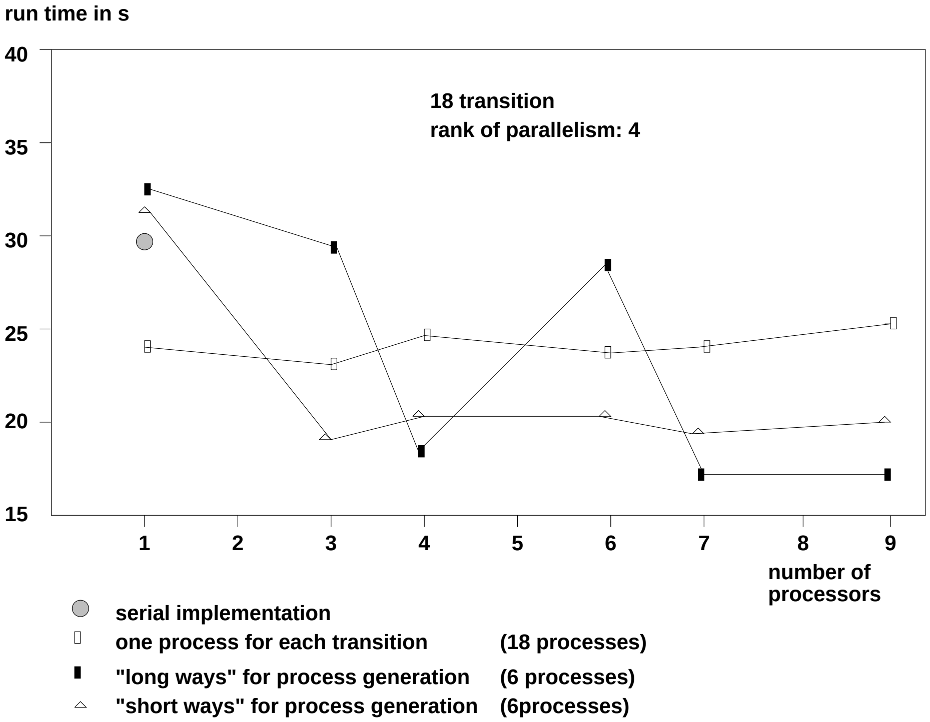

Figure 13: Time Behaviour of Parallel Programs

6.2 Load Balancing

In the last section we have divided the net into fixed processes in each case. Thus a clearly structure of processes and communication betweem them was generated.
Because input data dependencies very often influence the run time behaviour of a given solution, general performance predictions will be impossible or only useful for strong restrictions concerning the range of all (!) input data. Finally it needs to be mention that the reachabilty graph can be constructed only for small systems with respect to the necessary expenditure.

So it is usual to use (stochastic) results from a simulation process or to look for a dynamic adaption. While implementing a net a lot of useful results can also be obtained from the running program and can dynamically influence the execution of the running program. Because of the suggested implementation methods using Petri Nets as the basis of a graphical programming language there are special relations in the implemented software too and so often only a small communication and management overhead is needed for a change into a dynamic system. For doing so it is necessary that the whole net is present on each node although only a part of the net will be momently processed by this unit. In dependency of the use of direct or indirect methods related to the above definitions -2- or -3- for the implementation, the statistic about the firing processes will influence:

- the group of transitions which are assigned to the node (-3-)
- the path of the token flow, e.g. the structure of the actual state machine (note that in this case the system of state machines can be changed by time)

In the first case the statistic contains information about the number of tests of each transition on the node and the number of sucessful firing processes. So after some times it can be seen which transitions have caused the most work for the node. Comparing this load results with those of the other units after a longer period makes it possible to transfer the execution of code representing special transitions and a number of adjacent places to another node. Therefore only the marking of the places and a short name of the transitions must be sent, because all nodes contain the whole topology of the net. Note that for all management and communication processes time intervals can be used, in which the involved processing nodes do not require any communication. After passing all nodes twice in such intervals each node can decide which balancing actions it starts in order to improve the efficiancy in the next period.

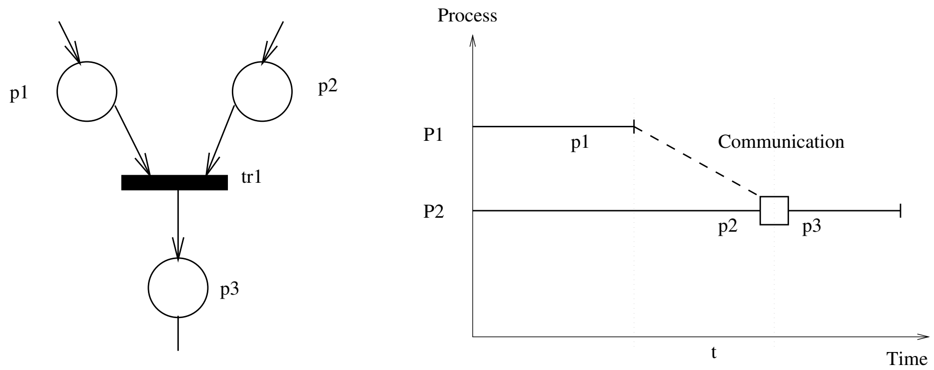

Figure 14: A System of two Processes

The second approach yields from a flexible interpretation of the elements of a special or transformed Petri Net, e.g. from the transformation described in the section above. The topology of the net has to be broadcast to all nodes too. The only difference is that each node gets a special information about the token, e.g. each node or process sees only one token and its movement. This token represents the token flow in one of the state machines which were directly derived from the topology of the net (as described above each state machine is a subgraph of the transformed net).
Now lets consider the situation when implementing the special structure of the element from figure 14. It is obvious, that 2 processes $P1$ and $P2$ will be generated. After some instruction for instance the process $P2$ needs some data generated by $P1$. When receiving this information the execution of the process $P2$ will be continued and $P1$ will be terminated. The same algorithm can be represented if process $P1$ continues the execution after receiving

some other information from process $P2$ which than finished its execution. A load balancing between the processors executing $P1$ and $P2$ could be achieved if the following part after $p3$ will be processed on that machine which has less to do than the other. This requires that

- each machine can execute each part of the work and
- all alternatives of the execution are included in the structure of an uniform process which can be multiple started with several entry points.

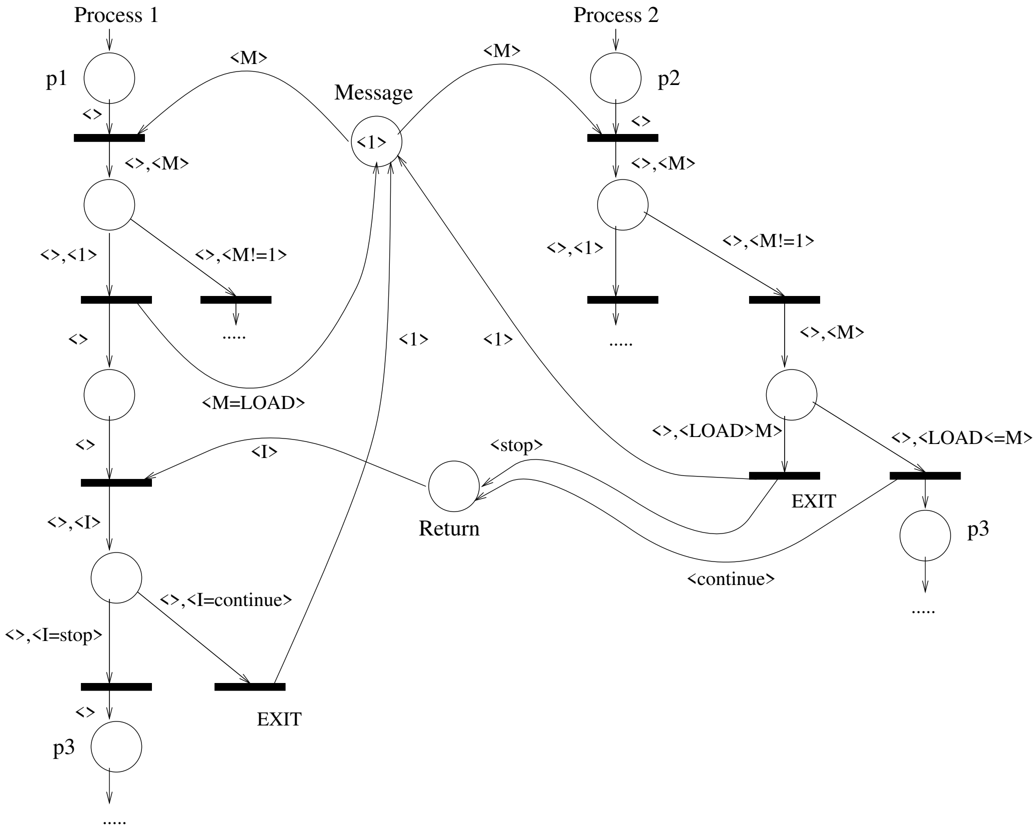

Figure 15: Introducing the Laod Balancing

Now we are ready to describe our load balancing approach. In common sense the required information will not be generated in time, in most cases one process has to wait a time ($_\Delta t$) for the communication service. Let us consider to processes with the above-mentioned structure. Without any restrictions to the universality lets suppose that $P1$ reachs the commmunication point at first. The time until $P2$ finished its work can be used for calculating an optimal load balancing strategy without influencing the execution time of the whole program [3]. Figure 15 shows the Petri Net model of the needed communication procedures, if $P2$ passes $p2$ at first all necessary structures can be symmetrically added.

[3]if there is a pure multiprocessing mode; in a multitasking mode the computing time can be neglacted, if the number of processes is $>> 1$

The functionality is the following. If $P1$ reads $Message$ at first it contain the initial date $< 1 >$. This will be replaced by the actual load factor of the corresponding CPU. Because $P1$ has to wait for $P2$ in each case no time will be wasted if the difference $_\Delta t$ is big enough. Passing the communication point $P2$ gets the information about the load factor of $P1$ and can compare this information with its own. So $P2$ can decide if the process will be continued on its machine or not. Furthermore $P1$ gets the decision via another communicaition and can continue its execution now. This communication is not a new overhead because $P2$ must communicate with $P1$ in each case, also without the described load balancing. Only the decision of $P2$ about the load balancing had to be added to all other information in the message. After all, the initial value of $Message$ will be restored for following requests by the exiting process.

In this case the decision was made from temporarily available load informations but other ones, e.g. waiting messages and so on can be used too. As it could be seen good possibilities for an adaptive load balancing can be derived.

7 Conclusions

In this report a software development approach is shown completely based on the efficient use of Petri Nets in each stage of work. This is an addition to the well known use of nets for modelling parallel and distributed architectures and software solutions. An important progress for a performance oriented design was achieved by the consideration of soft- and hardware structures in an uniform model. Furthermore it was exemplary shown for a basic class of Petri Nets that a part of such a complex model can be automatically transformed into an executable program. The generated solutions are more efficient than other comparable ones and contain a lot of possibilities for an optimization of the run time behaviour. The authors suggest for such an optimization the use of decentral and dynamic approaches based on run-time statistics or the use of runtime information about the load of the processors. Because of the special structures achieved from the transformation of nets the introduced methods achieve good results and can adapt all requierments resulting from input data dependencies.
In such way an evaluation of Petri Nets into a kind of graphical programming language for parallel and distributed systems can be expected.

References

[1] F. Breant: Rapid Prototyping From Petri Net on a loosly Coupled Parallel Architecture, Applications on Transputers 3, S. 644-649, IOS-Press, (1991)

[2] B. Daene, H. Unger: Problem-oriented Design of Parallel Programs in: Parallel Computing: Trends and Applications, p.597-600
(Proceedings of PARCO '93), G.R. Joubert, D. Trystram, F.J. Peters and D.J. Evans (Editors), Elsevier Science B.V., (1994)

[3] W. Fengler, I. Phillipow: Entwurf industrieller Microcomputersysteme, Carl-Hanser-Verlag, Muenchen, (1991)

[4] M. Heiner: Petri Net Based Software Validation, TR-92-022, ICSI, (1992)

[5] R. Knorr: Spezifikation, Verifikation, Leistungsbewertung und Implementierung von Kommunikationsprotokollen mit hierarchischen High-Level-Netzen,
TU Ilmenau, Department of Computer Science, Dissertation, (1994)

[6] U. Lichtblau: Graphtransformationen zur Erkennung adaaehnlicher Kontrollstrukturen, Dortmund, DFG-Forschungsbericht, (1980)

[7] W. Reisig: On a Class of Co-operating Sequential Processes,
1st. European Conference on parallel and distributed processing, Toulouse, (1979)

[8] W. Reisig, W. Brauer, G. Rozenberg: Petri nets: Applications and Relationships to other Models of Concurrency.
In: *LNCS 255*. Springer Verlag, Berlin-Heidelberg-New York, (1987)

[9] G. Hipper, M. Klein, D. Tavangarian: Parallel Computing in Workstation Clusters using a Concurrent Network Architecture, ISSM94, Washington, D.C., (1994)

[10] G.S. Thomas: Parallel Simulation of Petri Nets, Technical Report, University of Washington, (1991)

[11] H. Unger: Moeglichkeiten zur Implementierung von Petri-Netz-Modellen auf Mehrprozessorsystemen, Dissertation A, TU Ilmenau, (1994)

[12] H. Unger, K. Ben Achour: A method for the design of parallel programs for a multiprocessor system,
In: *LCNS 634*. Springer Verlag, Berlin-Heidelberg-New York, (1992)

[13] S. White, A. Alund, V.S. Sunderam: Performance of the NAS Parallel Benchmarks on PVM based Networks, NASA, RNR 94-008, (1994)

[14] J. Winkowsky: A Distributed Implementation of Petri Nets, ICS PAS-Report518, Polish Academy of Science, Warschau, (1983)