AN APPROACH DESIGNING PARALLEL SOFTWARE FOR DISTRIBUTED CONTROL SYSTEMS

H. Unger*, **B. Däne**** and **W. Fengler** **

**University of Rostock, Department of Informatics, D-18051 Rostock; Germany. E-mail: hunger@informatik.uni-rostock.de*

***Technical University of Ilmenau, Department of Informatics and Automation, D-98684 Ilmenau; Germany. E-mail: bdaene!wfengler@theoinf.tu-ilmenau.de*

Abstract. Petri Nets have been proved to be an effecient tool to represent complicated systems. Nevertheless, in general it is not easy to implement a technical system given as a Petri Net on a multiprocessor system. This contribution presents a new approach for this procedure. The main difference compared to other methods is the effective use of message passing communication during the implementation.

Key Words. Petri-nets; Distributed computer control systems; Parallel programs

1. INTRODUCTION

Progress in hardware design makes it possible to use multiprocessor architectures even in small automation systems. Parallel programming requires effective methods to find out parallel executable parts in a given algorithm (Boillat, *et al.*, 1991). Therefore it is necessary to solve a lot of problems in a transparent way for a wide group of users. That is why Petri Nets, a graphical language of description, became more and more important for modelling parallel software solutions (Reisig, *et al.*, 1987).

But there are only a few approaches for implementing Petri Net models on different multiprocessor architectures (Thomas, 1991; Unger, 1994). An overview is given in figure 1.

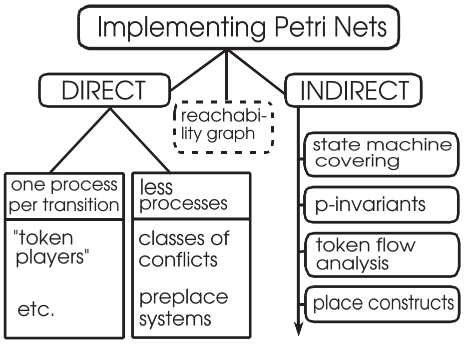

Fig. 1. An overview about existing implementation approaches

From the authors point of view all known methods fall into two basic types. The first one - the so called direct type - implements processes according to the transitions of the net. The second, indirect one is to decompose or to cover a given Petri Net by state machines, and then to implement one process for every state machine. Especially if a Petri Net has many transitions, the first method yields in each case a solution with plenty superfluous of processes and a large communication overhead. In general, the second group of methods generates a more efficient code, but, in contrast to the first one, it does not apply to all Petri Nets.

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] require a client server relation instead of a shared memory for solving this problem and so the number of parallel working processes is increased.

The present paper shows a new approach for an implementation avoiding the disadvantage described above.

[1] **P**arallel **V**irtual **M**achine for UNIX clusters from the Oak Ridge National Laboratory (Sunderam, 1990)

2. BASIC CONCEPTS

Usually 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) $V : F \rightarrow \mathbf{N}$, the multiplicity function

(iv) $m_0 : P \rightarrow \mathbf{N_0}$, the initial marking ($\mathbf{N}$ and $\mathbf{N_0}$ denote the sets of positive and nonnegative integers, respectively.)

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 $p \in P$.

Additionally is defined: $pF = \{t | (p, t) \in F\}$, $Fp = \{t | (t, p) \in F\}$, $tF = \{p | (t, p) \in F\}$ and $Ft = \{p | (p, t) \in F\}$.

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 (Fengler and Philippow, 1991):

(1.) $\mathbf{w_x}$,
a set of boolean expressions associated to 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 associated to 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 to the places of P.
Procedures are started when a new token reaches the place.

Implementing a given Petri Net means to transform it into a program by interpreting sets of elements as structures of a parallel program. When doing so, the state of the program or a class of its states can be derived from an actual marking and vice versa.

3. 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.

3.1. *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 2). All the structures consist of a set of transitions A and a set of preplaces S of the transitions of A in such a way that there is at least one transition to each other one which has a common preplace.

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

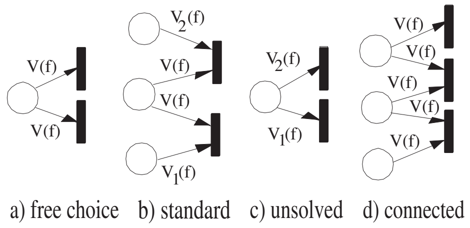

Fig. 2. Static Conflict Structures in a Petri Net

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:

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

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 (2d)) is represented in the maximal conflict set $K^k(\Pi, \Theta)$.
In order to get a set with all preplaces of $t \in \Theta$ in $(\Pi, \Theta) = K^k(\Pi_0, \Theta_0)$ the set system Π is modified by $\Pi' = \{p | \exists t \in \Theta : (p, t) \in F\}$.

For further transformation such structures (see figure 3) have to be cut out 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, when this transition has got a message from the prearea.

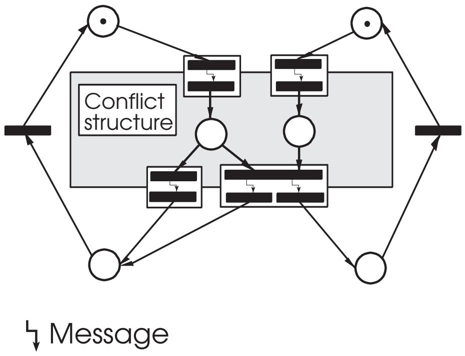

Fig. 3. Separation of Conflict Structures

A later discussion shows that only the more difficult conflict situation in figure (2c) must be cut out.

3.2. *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, an unmarked place construct $(P'(p), T'(p), F'(p), V'(p))$ is defined 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, one gets 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 is defined

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

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

$$P'(p) = \{p'_0, .., p'_i, x_1, .., x_e\}$$

with $i = 0(1)u$ and $e = 1(1)|t_{out}F|$.

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

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

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, for each $(a, b, c) \in C_2 \cup C_3$ arcs have to be added 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 4.

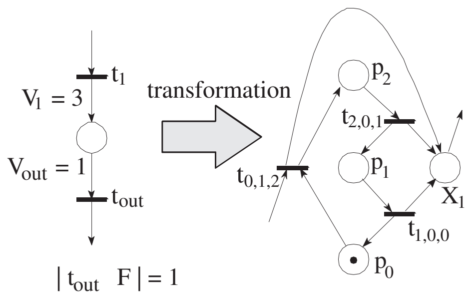

Fig. 4. Example of an Easy Place Construct

(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 for all $(a, b, c) \in C_1(v_e) \cup C_3(v_e)$ add an arc to F' with

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

(3.)
A marking m' of Φ' is said to be corresponding 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) $\forall i, j : if \quad m'(p'_i) = 1, p'_i \in P'(p)$
$i + m'(x_j(p)) * V_{out} = m(p)$

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

4. IMPLEMENTATION

Implementing Φ' means to find out interpretations for special elements of the given Petri Net. This work falls into two parts: implementing the conflict structures and implementing the transformed remaining Petri Net Φ'.

The main problem with implementing conflict structures results from the shared use of a data object representing places with more than one posttransition. The new 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.

Now 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 (Unger, 1992). 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. 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 in the transformed Petri Net. 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 5. Because the multiplicity of all arcs is equal 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 one gets a superposition of such process groups.

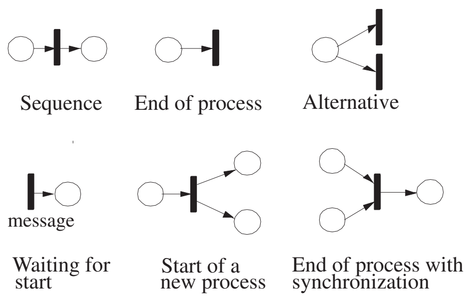

Fig. 5. Elements in the Reduced Transformed Petri Net

In all cases there is the restriction that in a given moment only one transition of each place construct can be fired. 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 special values 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, 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.

Results from a experimental implementation of a control program achieved by several methods are shown in figure 6.

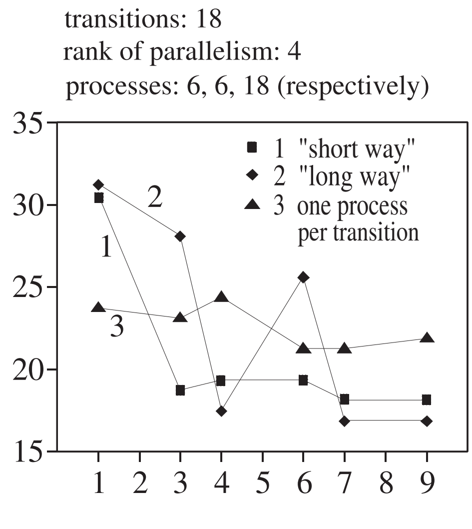

Fig. 6. Time Behaviour of a Parallel Program

5. CONCLUSION

A new method for the automatic generation of parallel software from Petri Nets has been shown and some transformation and implementation details have been discussed. The method enables the generation of more efficient parallel code by preventing some communication overhead resulting from conflict situations in the net. A first experimental implementation has shown the expected results.

6. REFERENCES

Boillat, J. E. *et al.* (1991). *Parallel Computing in the 1990's.* Institut für Informatik, Universität Basel.

Fengler, W. and I. Philippow (1991). *Entwurf industrieller Mikrocomputersysteme.* Carl Hanser Verlag, München-Wien.

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

Sunderam, V.S. (1990). Parallel Virtual Machine. In: *Concurrency: Praktice and Experience,* **No. 12,** 315-339.

Thomas, G.S. (1991). *Parallel Simulation of Petri Nets.* Technical report, University of Washington.

Unger, H. (1992). A Petri Net Based Method to the Design of Parallel Programs for a Multiprocessor System. In: *LNCS 634.* Springer Verlag, Berlin-Heidelberg-New York.

Unger, H. (1994). *Untersuchungen zur Implementierung von Petri-Netz-Modellen auf Mehrprozessorsystemen.* Dissertation, TU Ilmenau.