A Model of Processes Based on Petri Nets

K.P. Gostelow*
Department of Information and Computer Science
University of California
Irvine, California

T.J. van Weert**
Lerarenopleiding "Ubbo Emmius"
Sectie Wiskunde, Postbus 2056
Groningen, The Netherlands

ABSTRACT

This paper describes the Set Model of processes which was developed to help put the design of process-oriented systems on a sound footing. The basis of the Set Model is that of a process description: any sub-net of a Petri net, such that, the sub-net is itself a Petri net. The real strength of the definition appears when one realizes that Petri nets and Petri sub-nets are defined as sets. Thus, process descriptions are related just as their set descriptions are related (for example, sharing is set intersection), and set theory is used to operate on process descriptions. The definition may be applied recursively, in that the Petri net processor (the process which interprets nets in Petri net language by firing transitions and moving tokens) of a given Petri net may itself be described by a Petri net. Thus the Set Model can represent micro-programmed machines and multi-level interpreters. As an example, the model is used to describe a procedure whereby any number of producer and/or consumer processes can construct a channel for interprocess communication. The technique allows parallel and independently operating channels to be constructed. In contrast to other models of processes and their communication, interprocess communication is not a postulated component of the model, but rather is constructed from more basic notions. Also the Set Model shows how Petri nets can be used to model the dynamics of systems, such as the creation and destruction of processes and intermodule linkage.

1. Introduction

Processes and interprocess communication are concepts which are used heavily in the design of computing systems. Yet the literature contains little on the fundamental aspects of these notions. In particular, we are interested in developing a system for interprocess communication which is based on sound fundamental principles in the hope of simplifying and unifying the design of computer systems and networks. Using Petri nets as a basis, we developed the Set Model of processes which we believe captures

*The work reported here was done while at the Stichting Academisch Rekencentrum Amsterdam, Amsterdam, The Netherlands.

**The work reported here was done while at the Rekencentrum Rijksuniversiteit Groningen, Groningen, The Netherlands.
the general notion of process in a very concise and useful manner. The Set Model begins by defining a process description as any subset of the places, transitions, and tokens of a Petri net, such that, the subset is itself a Petri net. This characterization of a process description implies several properties one usually ascribes to a process; for example, its actions affect only those things within its domain. Thus, one process can directly affect another process only if their domains "share", i.e., their process descriptions contain a common element. This definition of process description allows us to represent sharing, (multi-level) processors, interprocess communication, and other concepts within the model. It is also possible to model the creation of process descriptions, the linking together of process descriptions, and the dynamics of the construction of paths to be used for interprocess communication.

2. The Set Model

2.1 Process descriptions and Petri nets

The model is based upon Petri nets, where we say that $P = (\text{PLACES}_P, \text{TRANSITIONS}_P, \text{TOKENS}_P)$ is a Petri net if

(1) $\text{PLACES}_P$ is a finite indexing set of elements called places

(2) $\text{TRANSITIONS}_P$ is a finite indexing set of elements called transitions, where for each $t \in \text{TRANSITIONS}_P$
   (a) $t$ may reference any number of input places $p_i$ and any number of output places $p_o$, so long as each such $p_i$ and $p_o$ are in $\text{PLACES}_P$ (referencing is represented by directed arrows).
   (b) there is at least one input place to $t$.

(3) $\text{TOKENS}_P$ is a vector specification (called the marking) of the number ($\geq 0$) of tokens on each place $p \in \text{PLACES}_P$.

Since $\text{PLACES}_P$ in net $P$ is an indexing set, if the marking is $q$ then $q_p$ is the number of tokens on place $p$. If $P = (\text{PLACES}_P, \text{TRANSITIONS}_P, \text{TOKENS}_P)$ and $Q = (\text{PLACES}_Q, \text{TRANSITIONS}_Q, \text{TOKENS}_Q)$, where $\text{PLACES}_Q \subseteq \text{PLACES}_P$, $\text{TRANSITIONS}_Q \subseteq \text{TRANSITIONS}_P$, $(\text{TOKENS}_Q)_r = (\text{TOKENS}_P)_r$ for all $r \in \text{PLACES}_Q$, then we write $Q \leq P$. Finally, if $P$ is a Petri net, then $Q$ is a Petri sub-net.
of $P$ if $Q \subseteq P$ and $Q$ is itself a Petri net. For example, Figure 1 shows a Petri net $P$ and four Petri sub-nets of $P$ called $P$, $Q$, $R$, and $S$. $T$ is not a Petri sub-net of $P$ because the transition $t$ in $T$ references a place $g$, but $g$ is not a part of $T$; this violates condition (2a) of the definition of Petri net.

**Theorem:** Let $P$ be a Petri net and $Q \subseteq P$. Then $Q$ is a Petri sub-net of $P$ if

$$(\forall t \in \text{TRANSITIONS}_Q) \ (t \text{ references place } p \Rightarrow p \in \text{PLACES}_Q)$$

**Proof:** Conditions (1) and (3) of the Petri net definition are immediate. Condition (2a) is assumed as hypothesis of the theorem. To show that condition (2b) is satisfied in $Q$, simply note that every $t \in \text{TRANSITIONS}_Q$ is also in $\text{TRANSITIONS}_P$ and that (2b) is satisfied in $P$. □

A process description is defined to be a Petri sub-net. Thus in Figure 1, those systems circumscribed and identified as $P$, $Q$, $R$, and $S$ are process descriptions, while $T$ is not a process description. $R$ is a special case and is the empty process description; $S$ is also a special case in that it contains no transitions. (You might prefer to consider a process description such as $S$ a segment.)

A given Petri net is a syntactically valid statement in the Petri net language, the semantics of which are given by the actions of the postulated Petri net processor on that statement. The Petri Net Processor (PNP) operates by arbitrarily selecting one transition $t$ and testing to see if all input places to $t$ contain at least one token. If so, then that transition is enabled and may fire. Firing consists of removing one token from each input place of $t$ and placing one token on each output place of $t$. The activity resulting from the interpretation of a process description (Petri sub-net) by the Petri net processor is defined to be a process. We will, however, be concerned with processes per se, but rather only with their descriptions as
Petri nets since structural relationships are the object of the investigation; topics concerning flow are considered in [1,4].

2.2 Sharing among process descriptions

Figure 1 shows (among other things) that process descriptions S and Q share place c. In Figure 2 processes P, C₁, and C₂ all share places e and f which comprise all of Q. Figure 2 is a representation of the producer-consumer bounded-buffer InterProcess Communication (IPC) mechanism of Dijkstra [2] and Habermann [3] operating in a one-producer two-consumer system. Two process descriptions A and B are defined to share if \( \text{PLACES}_A \cap \text{PLACES}_B \neq \emptyset \) or \( \text{TRANSITIONS}_A \cap \text{TRANSITIONS}_B \neq \emptyset \). That is, sharing implies a non-null intersection of the set descriptions of the processes.

Note that in order for direct IPC to take place between two processes, it is necessary that their process descriptions share in some fashion. This is a consequence of the particular definition of a process description which we have adopted. A process description is a self-contained Petri net; it is a "closed" system. Hence for process A to directly affect (directly communicate with) process B, the process descriptions of A and B must contain common elements - they must share.

3. Modeling the Petri net processor PNP

A process was defined to be that which resulted from the application of a processor (the Petri net processor) to a process description (a Petri subnet). In this section we show that for any given process description, the PNP interpreting that process description may be described by a Petri net.

Figure 3a shows a simple process description P, while Figure 3b shows a particular higher-level encoded form of P. The encoding is done by representing each transition of Figure 3a by two rows of places in Figure 3b: the first of the two rows specifying the input places of the encoded transition,
and the second of the two rows specifying the output places. A place and
the tokens on it in the Petri net of Figure 3a are represented, in Figure 3b,
by a place with the same number of tokens. In Figure 3b, each pair of
places in each of the two rows representing a transition is essentially a
boolean, stating whether or not the corresponding place is referenced by that
transition. Figures 3a and 3b each give the same information, but the process
description P from Figure 3b may be interpreted by a Petri net emulating the
PNP, whereas the process description P from Figure 3a is to be interpreted by
the PNP itself.

Figure 4 shows an implementation of one "place-slice" of PNP as a
Petri net. To construct an entire Petri net processor, there must be at
least as many place-slices of the processor as there are places in the net
to be interpreted. The place-slice of the processor shown in Figure 4 is
operating on place a of process description P. For the reader's convenience,
the entire process description P is shown (in its encoded form) with the places
holding the description of P given as two concentric circles. Also, some
notational shorthand has been used: multi-headed arrows indicate references
by more than one transition to the same place, or to more than one place by
the same transition; double-headed arrows indicate a reference to a place
as both input and output of that transition. The processor operates by
selecting the first transition to interpret, shown as transition s in
Figure 4. The input-place description of transition s (the top row of places)
is then copied into the "transition input register" of the processor, and the
output-place description is copied into the "transition output register" of the
processor. After copying, the basic firing rule is evaluated: \( f_1 \) in the
processor removes a token from place a of process description P if the
transition s being interpreted references place a as an input place, and there
is at least one token on place a; or $f_2$ in the processor allows the "a" portion of s to fire if s does not reference place a as an input place; or $f_3$ in the processor decides not to fire the "a" portion of s when a is referenced as an input place but there is no token on place a. The individual place-slice firing decisions are then all brought together and synchronized by the two transitions "fire decision" or "cancel decision" in the PNP emulator. Only if every place-slice decides to fire or not to fire the transition currently in the transition register, is the decision action taken. If any one place-slice cannot fire, then the others wishing to fire must cancel via transitions "cancel 1" or "cancel 2" in the PNP description. If firing is to be done, the input places have already been decremented, and the output places are then incremented by $g_1$ (or not incremented by $g_2$) for each place-slice. The transition registers are then cleared. After each place-slice in the transition registers has been cleared, the instruction-counter (places s, t in Figure 4) is incremented and the next transition to be interpreted is selected.

**Theorem:** For any process description P, there is a PNP for P which may be given as a process description.

**Proof:** Given by the construction above and the observation that the resulting Petri net description of the PNP emulator is also a process description.$\square$

The converse is not true, since a given Petri net implementation of PNP has a finite number of places and thus has a bound on the number of place-slices it contains. For any such bound, one can always construct a net to be interpreted which is larger in the number of places it uses than the capacity of the PNP emulator (the number of place-slices in the PNP emulator). Thus, there does not exist a universal PNP.
Corollary: For any given process description \( P \), \( \text{PNP} \) is a process.

By recursive application of the above theorem, one can see that the set model can model multi-level processors (e.g., micro-programmed machines) by encoding \( \text{PNP} \) and constructing another \( \text{PNP} \) to interpret the encoded \( \text{PNP} \), etc. Figure 5 shows the structural sharing relationships among a two-level processor system, where \( \text{PNP}_2 \) interprets \( \text{PNP}_1 \) which interprets \( P \). Note that \( P \subseteq \text{PNP}_1 \subseteq \text{PNP}_2 \) and that a process description is completely shared by its processor. Such a sharing relationship is necessary in order for a processor to access the necessary components in the description of the process it is interpreting. Finally, we note that we are not really restricted to the Petri net language and Petri net processors; other languages and their processors may also be modeled. (Petri nets cannot model all languages \([4,5]\), however, various extensions to Petri nets have been made \([5]\) for which the same theorem, as well as the converse, may be stated. These extended nets cover all Turing-computable languages.)

4. Operations on process descriptions

4.1 Union and difference operations

In this section we define two operations on process descriptions which may alter the structure of the sharing relationships among processes. Let \( P \) and \( Q \) be process descriptions. Then we define the union process description \( S = P \cup Q \) to be given by

\[
\begin{align*}
\text{PLACES}_S &= \text{PLACES}_P \cup \text{PLACES}_Q, \\
\text{TRANSITIONS}_S &= \text{TRANSITIONS}_P \cup \text{TRANSITIONS}_Q, \\
(TOKENS_S)_r &= (TOKENS_P)_r \text{ if } r \in \text{PLACES}_P \text{ or } (TOKENS_Q)_r \text{ if } r \in \text{PLACES}_Q.
\end{align*}
\]

Figure 6b is an example of the union of the two process descriptions of Figure 6a. Also, \( T = S - Q \) is defined to be the difference process description and is defined to exist only if: for every transition \( t \in \text{TRANSITIONS}_S \setminus \text{TRANSITIONS}_Q \) and place \( p \) such that \( t \) references place \( p \), then \( p \in \text{PLACES}_S \setminus \text{PLACES}_Q \). It \( T = S - Q \).
exists, then \( \text{PLACES}_T = \text{PLACES}_S - \text{PLACES}_Q \), \( \text{TRANSITIONS}_T = \text{TRANSITIONS}_S - \text{TRANSITIONS}_Q \), \( (\text{TOKENS}_T)_r = (\text{TOKENS}_S)_r \), for all \( r \in \text{PLACES}_T \). Figure 6c shows the result of the difference operation \( T = S - Q \) where \( S \) and \( Q \) are from Figure 6b.

**Theorem:** If \( A \) and \( B \) are process descriptions, then \( A \cup B \) and \( A - B \) (if it exists) are process descriptions.

**Proof:** \( A \cup B \) - Conditions (1), (2), and (3) are immediate.

\( A - B \) - Conditions (1) and (3) are immediate. Condition (2a) holds since the requirement for the existence of \( A - B \) is that if \( t \) is a transition in \( A - B \) and \( t \) references place \( p \), then \( p \) is a place in \( A - B \). Condition (2b) holds for \( A - B \) because it holds for \( A \). \( \square \)

### 4.2 Create and destroy operations

Let a process \( S \) consider some part of itself, say \( P \), to be a subprocess of \( S \); then in the Set Model, \( P \) is a process description with respect to \( S \) so long as \( P \) satisfies the definition of a process description. Any time some process \( S \) recognizes some part of its own process description as another process description \( P \), then we say \( S \) *creates* \( P \). Conversely, when \( S \) no longer recognizes \( P \) we say \( S \) *destroys* \( P \). Although we will ignore the details here, it is possible to construct Petri nets which describe the creation and destruction of subprocess descriptions inside themselves, and we will assume processes to possess such a capability where needed (Section 5).

### 4.3 Dynamic modification of processes

Consider the process description \( P \) in Figure 3a, and its encoded form in Figure 3b. We constructed one particular Petri net to operate on \( P \) in Figure 3b to emulate PNP. Other nets operating on \( P \) could be constructed as well. For example, in Figure 7a we show \( P \) being operated on by process \( T \). When transition \( w \) in \( T \) fires, place \( b \) in \( P \) is no longer specified as an output.
place of transition \( t \) in \( P \). Figure 7b shows the equivalent modification performed on \( P \) in Petri net language.

Clearly, any desired modification of the reference specifications of the transitions of a net can be made in such a fashion. However, one must beware of modifying a net to the point that it is no longer a Petri net (for example, by specifying a transition to have no input places). Just as a person can construct illegal nets, a process described in a Petri net can construct other illegal nets.

5. An example: interprocess communication

The problem is to describe a procedure whereby any number of mutually interested processes, acting as producers and/or consumers, can construct a path (channel) for interprocess communication. We say a path has been constructed when the processes concerned share and access a bounded-buffer as in Figure 2. (Mechanisms other than the bounded-buffer could be selected to demonstrate the fact of IPC, however, we feel that the bounded-buffer is a convenient, well-understood, and fundamental form of IPC and as such is an appropriate goal.)

Let \( A \) and \( B \) be two subprocesses created by a process \( OS \) (Figure 8a). Thus \( A \subseteq OS \) and \( B \subseteq OS \), and for non-empty \( A \) and \( B \) it is clear that \( A \) completely shares with \( OS \) and \( B \) completely shares with \( OS \). This is true for any creator/created pair, and represents an implicit ability for the creator and created processes to communicate through (some part of) that shared area. Hence, we simply observe that any creator (\( OS \)) and the processes created by him (\( A,B \)) can communicate if they so desire. In Figure 8a, each cross-hatch area represents the omitted Petri net details of a unidirectional IPC channel (such as a bounded-buffer); these channels are used by \( A \) to communicate with
OS, and by B to communicate with OS. The problem is for A and B to construct an IPC channel Q between themselves knowing only the name Q in common.

A process becomes connected to an IPC channel by performing a "setup" [6] on that channel. In the set model, process A is setup on channel Q by sending a message to OS requesting that OS perform the following:

1. create process description Q
2. A + A u Q
3. link Q to A (according to whatever convention has been designed into the system for IPC; here we have assumed the bounded-buffer mechanism).

Each of these steps has been defined and demonstrated above. Figures 8b-8d show the results of each of the three steps. Figure 8e shows the result of the setup by process B on Q; that is, steps (1)-(3) above are performed with "B" in place of "A", and "Q" is recognized as the same process for both A and B by OS. (Note that the communication between a creator and a created process may take place over an IPC channel such as we just described for use by A and B. This channel may be constructed by the creator (OS) at the time of creation of any sub-process (A,B) by unilateral setups by the creator on some agreed upon channel in their shared area.)

Thus the goal has been reached: an IPC channel Q has been constructed between A and B. By reversing the above steps, a process may detach itself from the channel. Also, simply by knowing different channel names, sets of producers and consumers may construct disjoint channels for IPC and carry out their conversations independently and in parallel with other conversations. Note that after the initial setup operation, no effort (overhead) on the part of OS is required to control message traffic.
6. Conclusions

The basis of the Set Model of processes is that of a process description: any sub-net of a Petri net, such that, the sub-net is itself a Petri net. The real strength of the definition appears when one realizes that Petri nets and Petri sub-nets are defined as sets. Thus, process descriptions are related as their set descriptions are related (for example, sharing is set intersection). The definition may be applied recursively, in that the Petri net processor (the process which interprets nets in Petri net language by firing transitions and moving tokens) of a given Petri net may itself be described by a Petri net. Thus the Set Model can represent micro-programmed machines and multi-level interpreters.

The set theoretical operations of union and difference of process descriptions were defined, as were the notions of create process and destroy process, and the linking together of processes by altering their descriptions. These operations were then used to describe a procedure whereby any number of producer and/or consumer processes could construct a channel for interprocess communication. The technique allows parallel and independently operating conversation channels to be constructed with no system overhead for message transaction. Also, note that in contrast to other models of processes and communication, interprocess communication was not a postulated component of the model, but rather was derived from a more basic notion: sharing.

7. References


P, Q, R, and S are process descriptions (Petri sub-nets); T is not a process description.
Figure 2

P is a producer, C₁ and C₂ are consumers,
Q is the bounded-buffer
Figure 3a
Process description P

Figure 3b
Encoded form of process description P
A "place-slice" of a PNP emulator...

Figure 4
Figure 5
Sharing relationship among processes in a two-level processor system
Figure 6a
Process descriptions P and Q

Figure 6b
S = P u Q
Figure 7a
T modifies P by firing w

Figure 7b
The modification of P in Petri net language
Figure 8a
A and B are subprocesses of and communicate with OS

Figure 8b
Q is created by OS
Figure 8c
$A + A \cup Q$

Figure 8d
$Q$ is linked to $A$
Figure 8e
B's setup on channel Q