A High-level Petri Net for Digital Systems

Wagner Luiz Alves de Oliveira
DSIF/FEEC/UNICAMP
Caixa Postal 6101
13083-970 Campinas, SP - Brazil
wlao@dsif.fee.unicamp.br

Norian Marranghello
DCCE/IBILCE/UNESP
Rua Cristóvão Colombo, 2265
15054-000 S.J. Rio Preto, SP - Brazil
norian@dcce.ibilce.unesp.br

Abstract

As a language for system modeling Petri net and its several extensions achieve a wide number of applications, such as: industrial process control, communication protocols, programming languages, distributed systems, computer architectures, and so on. These extensions enable the presentation of highly parallel systems, which can be simulated at the temporal, logical, and functional levels with reasonable simplicity. Even so, such extensions present some limitations when describing digital systems. The goal of this paper is to present an extension which is being developed to overcome such limitations, regarding the desirable characteristics of a modeling language for digital systems.

2. Proposal of a Petri Net Extension

Besides dynamic and structural analysis techniques, Petri net and their extensions present most of the features needed for a good methodology for digital system modeling. These features as stated by Gajski et al. [2] are: concurrency, state transitions, hierarchy, programming constructs, behavioral completion, exception handling, timing, communication, and process synchronization.

Place Chart Nets (PCN [3]) and Petri Nets for Embedded Systems (PNES [4]) are Petri net extensions that present most of the features stated above. PCN is a low-level hierarchical extension, with preemptive mechanisms, but it lacks timing and programming constructs. PNES is an extension with programming constructs, hierarchy and preemptive mechanisms, but cannot be classified as high-level Petri net due to their boolean tokens, each of which can either be associated to holding of a circuit’s behavior or not. However, as a language should make easy the system modeling process, Petri net tokens might carry complex data types, rather than boolean data types, to augment the descriptive power of the model. Also computation and communication time have not been simultaneously treated in Petri net extensions.

A high-level Petri net extension for digital systems (HPDS for short) is being developed. The main features of HPDS are (see fig. 1):

- **Places** describe modes of a modeled system (behaviors/local states). Places may receive **tokens** representing the functional or storage unit responsible for the mode described by the place.

- **Transitions** describe events that may occur in the system. Events are changes of modes. The effect of an event is represented by the elimination/creation of tokens from/to the places. The elimination of one token from one place represents the outcome of the mode (place) that was performed by system’s component (token). On the other hand, the creation of one token to one place denotes the beginning of the mode (described by the place) that will be performed by the system’s component.

- To each transition there may be associated:
  - **operations** to be done on the net variables;
  - a boolean expression, named **guard**, whose absence default to true; and

1. Introduction

The design of digital systems has reached a high degree of complexity that prevents their effective realization without CAD tools. Several specification languages have already been employed in such tools, each one with the objective of capturing as much hardware characteristics as possible.

Petri nets are powerful as a modeling language for many kinds of systems. Recently, Petri nets have shown to be an effective modeling methodology for discrete events systems, due to the existence of a set of Petri net related techniques for dynamic and structural analysis. These techniques allow the formal validation of more important properties in the model, such as: liveness (deadlock-freeness), conflict-freeness, and determinism, among others. An overview about the use of Petri nets in digital systems design is provided by Marranghello [1].

None of the Petri nets known to us meets all desirable characteristics of a good modeling language for digital systems. To overcome those limitations we are developing a Petri net extension whose specification is described in section two. In section three the features for a good modeling language for digital systems are provided, and their relationship with the proposed extension are made. A modeling example is given in section four. In section five we present some conclusions and steps to be taken towards the development of a Petri net based approach for description, simulation, analysis, and high-level synthesis of digital systems.
Each HPDS place has a **domain** representing the set of tokens allowed to be there. As a place is denoted by a LPDS, there is one instance of such sub-net for each token belonging to an HPDS place at any given moment.

- Conditions for **enabling** a transition are:
  - its guard must evaluate to *true*;
  - each input place connected to it through a simple or conjunctive branch, must have at least one set of available tokens that matches the removing expression of the branch;
  - each input place connected to it through a preemptive or interruptive branch, must have at least one token (available or not); and
  - each output place connected to it through a simple branch, must not have any token belonging to the token set resulting from the evaluation of the insertion function of the branch.

- The **firing rule** that must be satisfied by an enabled transition is that the lower time limit of the candidate transition must be lower than or equal to the lowest among the upper time limits of all enabled transitions.

- The **effect** of a transition firing is:
  - elimination of tokens from each input place and the creation of tokens in each output place, according to respective branches of the transition;
  - suspension of modes relating to places connected to the transition through interruptive branches;
  - resuming of modes relating to places connected to the transition through restore branches;
  - setting of the transition operations’ results; and
  - updating of the net times.

Four kinds of branches may connect places to transitions:
- **Simple**: it allows the elimination of a certain token set from the corresponding place in agreement with a *removing expression*;
- **Conjunctive**: it allows the elimination of a certain token set from each and every place belonging to group of connected places, also in agreement with a removing expression;
- **Preemptive**: it allows the elimination of all tokens from the corresponding place, independently these tokens are available for elimination or not; and
- **Interruptive**: it allows the suspension of the mode described by corresponding place.

Two kinds of branches may connect transitions to places:
- **Simple**: it allows the transition to put into corresponding place the token set described by an *insertion function*; and
- **Restore**: it allows the transition to continue the mode described by corresponding place that has been suspended by the firing of another transition connected to the same place through an interruptive input branch.

- Each place is **hierarchical** and is described by a low-level Petri net for digital systems (LPDS for short). LPDS is a Condition/Event Petri Net [5] that was extended with hierarchy, programming constructs and sojourn times for tokens at places.

- **a firing time interval** indicating the minimum and maximum delays for the actual firing of the transition after its enabling, whose absence default to occasional firing.

- Fig. 1: Half-duplex protocol HPDS model
3. HPDS and Representational Characteristics for Digital System

In this section it is shown how HPDS meets the features needed for a good methodology for digital systems modeling, according to Gajski et al. [2].

a) Concurrency

HPDS models can represent both data-driven (fig. 2.a) and control-driven concurrency (fig. 2.b).

b) State transitions

Like any Petri net extension, HPDS is built around the concept of states and transitions between them. Its places represent either local states or local behaviors, each one is treated as a computation mode. The change of modes take effect by firing the corresponding transition. Such firing represent the occurrence of an event in the modeled system under certain conditions.

c) Hierarchy

Some high-level Petri net extensions may include hierarchical constructs, by allowing either a place or a transition to represent a sub-net of the lower level of hierarchy. HPDS uses place related hierarchical constructs due to the following reasons:

- The places represent modes (states or behaviors) that can be held (states) or executed (behaviors) by one or more functional or storage units (tokens). Hence, as one may need to model sub-states of a state or sub-behaviors of a behavior, it is natural the choice for hierarchical mechanisms assigned to places; and
- In Petri net theory, transitions are atomic, i.e., must not be interrupted after being begun. So, if hierarchical constructs are associated to transitions, all operations described by the transition must finish before the control flow is transferred due to the occurrence of an exception. However, an exception requires that a mode be interrupted immediately.

As a mode (place) can be performed concurrently by one or more system units (tokens), there is one hierarchical instance for each token in the place at a given moment. For example, fig. 3 shows one of the instances concerning the sub-net of place B3 in fig. 1. This sub-net is represented with a LPDS.

d) Programming constructs

Few Petri net extensions hold out this feature. Some extensions present programming constructs aggregated to places (as PNES [4]) while other extensions present them associated to transitions (as Coloured Petri Nets [6]).

In HPDS, programming constructs are associated to both places and transitions. The place related constructs are used to system inherent data manipulation while those related to transitions represent manipulation concerning the net model control flow.

The association of programming constructs to places is only possible in the lowest level of hierarchy, which is described by a LPDS. With this net extension one can assign to each place a programming statement and a corresponding execution time (its time-stamp). Therefore, a token held by a top level place can only be consumed after all operations (statements) of the corresponding LPDS are complete, except in cases where preemption or interrupt events occur, such cases will be addressed further.

The use of conditional, iterative, and assignment programming constructs which operate on net variables and are associated to transitions give them a great representational capability and make the control of the net easier.

e) Behavioral completion

Behavioral completion is both the ability of a behavior to indicate that it has finished (all its computations have been performed) and the ability of other behaviors to detect this fact.

As a mode (place) can be performed concurrently by one or more system units (tokens), there is one hierarchical instance for each token in the place at a given moment. For example, fig. 3 shows one of the instances concerning the sub-net of place B3 in fig. 1. This sub-net is represented with a LPDS.

d) Programming constructs

Few Petri net extensions hold out this feature. Some extensions present programming constructs aggregated to places (as PNES [4]) while other extensions present them associated to transitions (as Coloured Petri Nets [6]).

In HPDS, programming constructs are associated to both places and transitions. The place related constructs are used to system inherent data manipulation while those related to transitions represent manipulation concerning the net model control flow.

The association of programming constructs to places is only possible in the lowest level of hierarchy, which is described by a LPDS. With this net extension one can assign to each place a programming statement and a corresponding execution time (its time-stamp). Therefore, a token held by a top level place can only be consumed after all operations (statements) of the corresponding LPDS are complete, except in cases where preemption or interrupt events occur, such cases will be addressed further.

The use of conditional, iterative, and assignment programming constructs which operate on net variables and are associated to transitions give them a great representational capability and make the control of the net easier.

e) Behavioral completion

Behavioral completion is both the ability of a behavior to indicate that it has finished (all its computations have been performed) and the ability of other behaviors to detect this fact.

As a mode (place) can be performed concurrently by one or more system units (tokens), there is one hierarchical instance for each token in the place at a given moment. For example, fig. 3 shows one of the instances concerning the sub-net of place B3 in fig. 1. This sub-net is represented with a LPDS.

d) Programming constructs

Few Petri net extensions hold out this feature. Some extensions present programming constructs aggregated to places (as PNES [4]) while other extensions present them associated to transitions (as Coloured Petri Nets [6]).

In HPDS, programming constructs are associated to both places and transitions. The place related constructs are used to system inherent data manipulation while those related to transitions represent manipulation concerning the net model control flow.

The association of programming constructs to places is only possible in the lowest level of hierarchy, which is described by a LPDS. With this net extension one can assign to each place a programming statement and a corresponding execution time (its time-stamp). Therefore, a token held by a top level place can only be consumed after all operations (statements) of the corresponding LPDS are complete, except in cases where preemption or interrupt events occur, such cases will be addressed further.

The use of conditional, iterative, and assignment programming constructs which operate on net variables and are associated to transitions give them a great representational capability and make the control of the net easier.

e) Behavioral completion

Behavioral completion is both the ability of a behavior to indicate that it has finished (all its computations have been performed) and the ability of other behaviors to detect this fact.
In HPDS, behaviors are described by places, each of which is explicitly or implicitly represented at the lowest hierarchical level by a LPDS. All LPDS net is composed by two boolean places, called begin and end, interlinked through a set of transitions and places. To each LPDS place there can be assigned one programming statement and its respective execution time.

When one token is created in a top level place, one boolean token is also created at the place begin belonging to the LPDS instance related to the token and place at the top level. At this moment, the top level token is said to be unavailable; this token will be available when a boolean token is created at the place end of corresponding LPDS instance. In other words, while the programming statements at the lowest level are not complete, the corresponding token at the top level will not be available for use.

The transitions connected to a place through a simple or a conjunctive input branch will not be able to eliminate unavailable tokens. Once available, tokens can be consumed by any enabled transition.

Transitions without firing time interval denote occasional events in the modeled system which may occur or not. On the other hand, transitions with firing time interval must fire, unless they become disabled due the firing of any other transition. A special case is the firing time interval \([0,0]\), that denotes immediate firing of the transition as soon as the necessary tokens for its enabling become available. This will occur after all needed behaviors are complete, i.e., when all respective input places receive the corresponding tokens.

**f) Exception handling**

In a modeled system, the occurrence of a particular event can require that a mode be interrupted or aborted immediately, transferring the net control flow for other mode. This kind of event is named exception.

Basically, there are two kinds of exceptions: **preemption** (when the current mode is aborted) and **interrupt** (when the current mode is temporarily suspended, can be resumed after a certain time).

The only Petri net extensions dealing exceptions are PCN [3] and PNES [4]. In PNES, a **supermode** is defined as a place related to a sub-net representative of operations that can be aborted by firing of a special transition called exception. A similar concept is found in PCN.

For the sake of preemption, HPDS makes use of so called preemptive input branch. This kind of branch allows the enabling of a connected transition whenever there is at least one token in the connected place, independently of such token to be available or not, i.e., of the place related mode be concluded or not. When the transition fires, all the tokens at the place will be removed, representing the abortion of the mode for each functional or storage unit. Fig. 4 shows the firing effect of transition Lose Packet (see fig. 1) on the instance of place B3 depicted in fig. 3. HPDS uses two kinds of branches to represent interrupt: interruptive input branch and restore output branch. The former branch suspends the mode held by a token, assigning false to the control variable **ENABLE**, associated with the transitions of the related LPDS instance. The restore branch have opposite action, assigning true to the mentioned control variable.

**g) Timing**

Most languages for digital systems modeling have no suitable representation of time. In the case of Petri nets, there are different mechanisms dealing with time according to extension employed. Some extensions assign time mechanisms to transitions while other assign them to places. However, no Petri net extension uses both approaches simultaneously.

Besides associating time intervals with the possibility of occurrence of transitions between modes (for modeling communication mechanisms) it has also been verified the need to assign times to modes (as PNES). In this way, HPDS works with two time structures: one that assigns time intervals to the transitions (denoting the minimum and the maximum delays for the firing of an enabled transition) and other that assigns time-stamps for the modes (denoting the delays for either behavior execution time or state holding).
h) Communication

Often, a digital system is composed by several behaviors that interact among them, in a cooperative way. It is important for a system description language to provide a generic communication that distinguish communication related behaviors from other system computation modes.

The HPDS transitions with firing time interval satisfy the communication needs explicitly since they allow that communication be performed within specific time constraints.

i) Process synchronization

Petri nets are asynchronous by definition. Nevertheless, the introduction of structural constructs for synchronization presents no problem. HPDS allows the use of control-dependent and data-dependent synchronization techniques.

Data-dependent synchronization can be achieved through the utilization of shared memory. In fig. 1, the net variable Count is used for inhibit the firing of transition Use Packets while the transition Store Packet and ID have not occurred for eight times.

One can obtain control-dependent synchronization by using transitions as synchronization points for computation modes such as happens with the transitions End Task 1 and End Task 2 at fig. 2.b. These transitions respectively return tasks T1 and T2 to the place Tasks, after concluding the related procedures (Proc A, Proc B, and Proc C for task T1, Proc C and Proc D for T2).

4. Modeling Example

Part of a HPDS model of the alternating bit protocol is shown in fig. 1. This protocol can be used to transmit messages between two entities allowing the transmission of one packet of information at a time, i.e., assuming half-duplex communication. Each message is divided in eight packets (which are represented by tokens p1 up to p8). It is assumed that packets or acknowledgements can be lost during the transmission process. Fault recovery is achieved through retransmission: the sender entity saves the time of transmission of a certain packet, and if the corresponding acknowledgement has not been received, the packet retransmission is done. This mechanism should also prevent the acceptance of packets that have already been received, i.e., the receiver entity must identify whether the received packet is a new packet or a packet that have already been processed. To solve this problem, before the transmission packets are numbered with sequential binary numbers; when a packet is received its identification number is returned to the sender by means of an acknowledgement packet.

In the example net model, there are places describing modes relating to either storage (B1, B2, B5, B6, B7) or processing (B3, B4, B8).

HPDS transitions describe events that may occur in the modeled system after completion of the corresponding enabling modes, which are described by the input places, with exception of cases where the places are connected to transitions through preemptive or interruptive input branches (when the completion of modes is not needed). In fig. 1, there are transitions for changes among modes (e.g., Receive Packet), preemptive transitions (Lose Packet and Lose ACK), one transition to begin an interruption (Interrupt), and one transition to return from an interruption (Resume).

Estimates of delay for each protocol action are required in the modeling process. In fig. 1, there are transitions with and transitions without firing time interval. The transition Send Packet is one of those that doesn’t have a time interval. So, the presence of only one packet (token) at place B1 is enough for enabling this transition, which must fire immediately.

The firing effect of transition Send Packet will be to remove one token (packet) from place B1 and the addition of new tokens to places B2 and B3. At this moment, transitions Lose Packet, Receive Packet, and Resend Packet are enabled, but only the first two can occur, since they have firing time interval [0,1], while the latter transition have firing time interval [5,6], denoting that a packet retransmission take place within a delay of at least 5 and at most 6 time units after the previous copy has been sent. Estimates within 0 and 1 time units are given for the rejection of packets as well as for the loss and the reception of both acknowledgements and packets. The sending of acknowledgements (transition Send ACK) have estimates within 0 and 2 time units.

Fig. 4: LPDS instance of place B3 after the firing of transition Lose Packet
Transitions Lose Packet and Lose ACK are connected to places B3 (Packet in Channel) and B8 (ACK in Channel) through preemptive input branches. As these places represent behaviors, their tokens will only be available after the corresponding processing has been concluded. If the computation relating to any of these places has to be aborted, preemptive branches will have to be used to enabling the transitions at any moment (as long as the other enabling conditions are satisfied). Fig. 3 shows the action of one preemptive branch over the LPDS representing place B3 (fig. 2) after the transition Lose Packet has occurred. Note the elimination of all tokens from the sub-net of place B3 (compare figures 2 and 3).

Another characteristic to be noted in fig. 1 is the use of conjunctive input branches, such as the one connecting places B5 and B7 to transition Use Packets. This branch is associated with a removing expression of tokens, denoting the class of tokens that may be simultaneously removed from both places B5 and B7. The removing expression $\{p1-p8\}$ states that the only token set that can be removed from the places B5 and B7 is the one composed by the tokens p1 up to p8. Therefore, the transition Use Packets will remove the eight available packets from B5 and B7. Note that Use Packets will just be enabled after the assignment of 8 to net variable Count, which will be reset if the transition fires.

Observe that the interruptive input branch of the transition Interrupt in fig. 1 will make unavailable the tokens at place B1, after its firing. This happens due to the impossibility of enabling the transition (or transitions) belonging to LPDS concerning place B1 (even if this subnet was not explicitly defined, by definition there always is a LPDS with one place begin and one place end, and at least one transition between them). On its turn, the restore output branch of transition Resume has inverse effect, not interfering with the enabling of transition(s) belonging to LPDS concerning place B1.

The simulation of a HPDS net is performed by changing classes. Each class describes a possible global state of the modeled system, and is composed by a net marking, the corresponding time domain, and the values of the net variables. The relationship among the classes is described by a reachability graph.

The properties of a HPDS model are responsible for validation or not of the properties of the corresponding actual system. The authors have already defined the following properties for HPDS [7]: boundedness (that enables the actual implementation of the modeled system), reinitiability (the system will return to initial state after finishing one or more tasks), and liveness (that asserts a deadlock-free system). After achieving the proper results, the net model of the system should be converted to a low-level Petri net in order to allow for a direct mapping of the net onto a circuit description.

5. Conclusions

We have described a Petri net extension called High-level Petri Net for Digital Systems (HPDS). The extension described here is the first step of the development of an approach for modeling, simulation and analysis of digital systems.

In this paper it was shown how HPDS includes features desirable to a modeling language for digital systems. Under the framework presented here the components, representational language, and analysis methods are being formalized.

The authors would like to thank professor Furio Damiani for several fruitful discussions during the course of this work and to acknowledge Oliveira’s doctoral scholarship under FAPESP grant 99/04321-7 that partially supported this research.

References