KRON: An Approach for the Integration of Petri Nets in Object Oriented Models of Discrete Event Systems

J.L. Villarroel, J.A. Bañares and P.R. Muro-Medrano

Departamento de Ingeniería Eléctrica e Informatica
University of Zaragoza
María de Luna 3, Zaragoza 50015, SPAIN

Abstract **This paper presents KRON, a representation schema to create models for discrete event systems. It is based in the integration of high level Petri nets with a frame based representation and an object oriented methodology. The main objectives considered in its definition are to obtain a comprehensive and powerful representation model for data and control, and to incorporate a powerful modelling methodology. KRON provides an efficient execution mechanism to make evolve the models. It is an adaptation of the RETE matching algorithm to deal with the features provided by high level Petri nets. Moreover, KRON allows the representation of decision points in the execution of non-deterministic models.**

1 Introduction

This paper deals with a representation schema to create software models for *discrete event systems* (DES). DES are dynamic systems that change its state at the occurrence of discrete events. In general, the state of *DES models* have logical or symbolic, rather than numerical values, and the events may also be described in non-numerical terms [14].

Complex DES are composed by elements which evolve concurrently and asynchronously. Elements interact each other by means of synchronizations and information passing mechanisms. There are several formalisms used to build models of complex DES, including those based on *Petri nets*, *calculus of communicating systems* and *communicating sequential processes*.

Petri nets have been recognized as a suitable mean to describe such complex DES. They allow formal analysis, graphic representations and the execution of the system models. Additionally, it is possible to fill the gap between the modelling and implementation phases by means of automatic code generation techniques (see for example [6]). However, the use of ordinary Petri nets in the modelling of large complex DESs can lead to models with unmanageable size. This drawback is reduced by using high level Petri nets [10] (e.g., coloured Petri nets or predicate/transition nets) which provide much more compact and manageable descriptions.

However, several researchers have established that the high level Petri net formalism is still lacking of two main aspects:

1. *Methodological aspects.* Several methodologies can be adopted during the system model design process using high level Petri nets. However, this formalism lacks of features to support explicitly concepts from modern systems engineering such as modularity, encapsulation, top-down and bottom-up designs, data abstraction, specialization, inheritance, ...
2. *Data representation aspects.* In the majority of practical applications, it is necessary to describe aspects that are not related with system dynamics. Moreover, in modelling complex systems (such as information or manufacturing systems), where there are complex relationships between elements, coloured tokens or tuples are not a powerful enough way of describing these relationships. To conclude, it is recognized that Petri nets are well process oriented formalisms, but not data oriented.

To deal with these deficiencies, the main approach adopted by the researches has been the integration of different paradigms in the same formalism. The objective in this case is to increase the high level Petri net formalism with other paradigms such as abstract data types, object oriented concepts or entity relationship models. There are a lot of integrated models in the technical literature, see for example: the Algebraic Nets [15], the Concurrent Object-Oriented Petri Nets [5], the OBJSA nets [3], the OONET (Object Oriented NET) [13], the EER formalism [7], the Object Petri Nets [11], PROTOB [1], HOOD nets [9]. However, the methodological and data representation aspects are not fully considered in the previous approaches. Most of the integrations are either data oriented or control oriented.

This paper illustrates the main concepts involved in KRON (Knowledge Representation Oriented Nets). The main objectives considered in its definition are: 1) to obtain an overall and powerful representation model for data and control; and 2) to incorporate a powerful modelling methodology. KRON is based in the integration of high level Petri nets with the following paradigms:

- *Frame based representation*: Frames has been selected as a basis to support the representation aspects due to its power to represent concepts and their relationships.
- *Object based methodology*: This approach offers methodological advantages such as: 1) supports conceptual models near to human conceptualization and

Figure 1: KRON inheritance class hierarchy.

independent from implementation, thus the models are easier to understand; 2) facilitates reusability and model extensibility based on encapsulation and inheritance characteristics.

Generally speaking, a KRON model is composed by objects, which can contain data, procedures and relationships. The frame/object-oriented approach undertakes the system modelling by explicitly describing its components and their interactions. In this way, the system model is structured as an image of the system itself. In addition to the features generally supported by object oriented languages, a set of semantic primitives implementing the high level Petri net formalism is included. The HLPNs provide the mechanism to describe the internal behaviour of the dynamic entities and the interactions between them.

The paper is organized as follows. First, a brief description of the modelling primitives and some methodological aspects are illustrated. The dynamic aspects and the model behaviour interpretation mechanism is described next. Finally, some conclusions from our experience are considered.

2 Modelling Primitives and Methodology of KRON

Following the object oriented approach, each entity in a KRON model is represented by an object which collects all relevant information about the entity and the relation-ships with other objects. These informations are structured in slots. The behaviour of dynamic entities is represented by means of a high level Petri net which is integrated with the rest of entity knowledge. Figure 1 shows the KRON inheritance class hierarchy that collects the more important concept specialization.

From the dynamic point of view (and directly integrating Petri Net features), there exists three classes of specialized objects:

- *Transitions* are objects which define the activities of dynamic entities that can produce state changes. They are equivalent to the transitions of a high level Petri net.
- *State objects* are used as the main objects representing the dynamic entities. They allow the access to all the information from an entity such as the information about the state and all other additional information such as physical description, relations with other entities, historical information, etc. The state is saved in a set of *place slots*. Each place slot corresponds to a single place of a high level Petri net. The activities that can produce a state change are saved in a set of *transition slots* in the state object. Each transition slot holds the transition object that represents the activity.
- *Tokens* are objects representing passive entities at the level of abstraction which is been considered. Tokens are the values located in the place slots and its distribution defines the system state. From the Petri net point of view they are the tokens evolving on the net.

There are also the so called relation objects which hold the information used to specify connections between KRON objects. KRON provides generic primitives to define relation objects and to establish relations between KRON objects. The most important predefined relations are:

- *Net relations* support the Petri net arcs and expressions labeling them, and are used to specify connections between places and transitions. The information about net relations is stored in transition objects.
- *Synchronization relations* provide a simple way to specify the interconnection between dynamic entities what is done by means of the synchronization of transitions. The firing of two related transitions is synchronized in a 'rendez vous'-like way. Both synchronized transitions are handled as one single object. In this way, communications between dynamic objects are supported by the same formalism that defines the internal dynamic of each object.
- *Objects relations* allow the definition of generic relations between objects. When a relation is established from one object to another, a slot is created in the first object with the name of the relation instance and an other slot is created in the second object with the name of the inverse relation. The updatings of direct-inverse relations are then automatic. The definition of composed colors from the high level Petri net point of view is possible by establishing some specialization of these relations.

To manage this representation schema there is an underlying modelling methodology in KRON. It can be split in the following steps:

1. Identification of the dynamic and no dynamic entities which compose the system to model.
2. Grouping of entities in classes.
3. Design a prototype for each class from the existing prototypes in the specialization hierarchy. If the entity to be modelled, at the actual level of abstraction, is passive it will inherit from the predefined *token object*, however if it is a dynamic entity its model will be composed by a *state object* and a set of *transition objects*.
4. Define the relations between classes. This activity is performed by object creation in a special hierarchy inheriting from the predefined *object relation*.

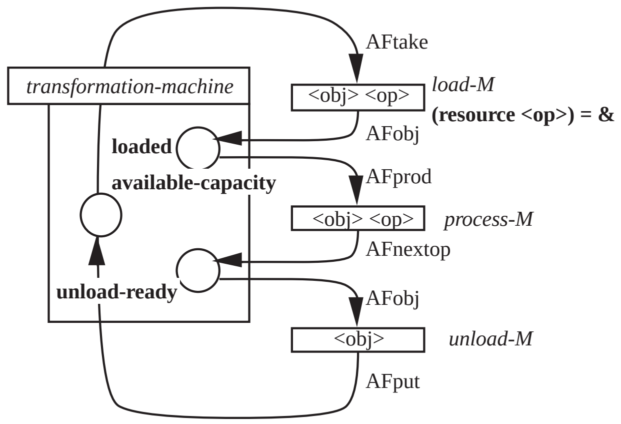

Figure 2: Manufacturing cell `C1`.

{*OP1*
is-a: operation
resource: M1, M3
next-operation: OP2
previous-operation:
product: P1
description: }

{*OP2*
is-a: operation
resource: M2
next-operation: END
previous-operation: OP1
product: P1
description: }

Figure 4: Operations and their relationships representing a process plan for workcell `C1`.

5. Obtain an instance to model each system entity from the corresponding prototype.
6. Representing the interactions between the system dynamic entities. It is performed by transition object synchronization.
7. Establishing of the initial system state (Petri net initial marking) by means of the assignment of token objects to place slots.

This modelling methodology follows a classical frame-object oriented approach. Only the two last steps are private of the proposed methodology. Transition synchronization is a more high level mechanism to communicate objects than the classical message passing. It allows to separate the model from the implementation and do not restrict the model to the client-server framework.

To illustrate the proposed representation schema and methodology, let us create the model of an unmanned manufacturing work cell. To simplify, it is considered a jobshop workcell `C1`, with three machines `M1, M2` and `M3`, a random access store `ST` and a pick-and-place robot `R`. Let us suppose that workcell `C1` can, in principle, manufacture two different types of products: `P1` and `P2`. The sequence of operations to be carried out on products of type `P1` is: `OP1; OP2`. Operation `OP1` can be carried out by machine `M1` or machine `M3`, while operation `OP2` must be carried out by machine `M2`. Products of type `P2` require the sequence of operations `OP3; OP4`. These operations are carried out by machines `M2` and `M3`, respectively.

It is easy to identify the entities involved in the system from the textual description of the example. These entities can be classified in: machines, stores and robots (dynamic entities); and parts and operations (passive entities). The prototypical models of these classes are then separately defined. Firstly, let us create the model of a typical manufacturing machine, which will be called `transformation-machine`. A `transformation-machine` model can be represented as an object (frame) containing information about its attributes, constitutive parts and structure. Characteristics such as abstraction hierarchy relations, predicted information and data collection, physical characteristics, interface features and graphic representation characteristics are also represented within the object definition.

In addition to the previous features, the dynamic behaviour must be represented. The behaviour of this physical entity is defined such as it can load parts to be processed and it has a limited available capacity. Three places (`available-capacity`, `loaded` and `unload-ready`) and three transitions (`load-M`, `process-M` and `unload-M`) are shown in figure 2. Places are identified in the `transformation-machine` textual representation (figure 3) by place slots and transitions by transition slots. The values of state slots identify the marking whereas the values of action slots are pointers to transition objects (the textual representation of transition objects `load-M`, `process-M` and `unload-M` is shown in the figure 4). Transitions in the model are associated with system actions used to change the state, which is itself represented by the net marking. Tokens can be identified in this case with the parts evolving in the cell. The presence of a token in a place is indicated by the presence of the frame name as a state slot value.

Machines `M1`, `M2` and `M3` can be defined as instances of the `transformation-machine` prototype. Following a similar method, `ST` and `R` must be defined as instances of isolated prototypes modelling a random access store and a robot. The instantiation process follows with the passive entities parts and operations. Figure 4 shows the textual representation of some operations involved in the cell. The model of operations defines several relationships. An operation is related with a resource (where it must be processed), with a product (the subject of process plan) and with other operations to represent a process plan. To illustrate how these relationships are defined, let us consider the order relation between operations. To establish this relationship, an instance of the specialized object relation *specification* is created. It defines the domain and range of the relation which are both operations. This object, sending to it an installation message, will create the suitable slots (*next-operation* and *previous-operation*) and mechanisms (demons) in the prototype of an operation to support the relation. In the instances of operations, only one of these slots must be specified because the other one will be automatically updated.

The structural description of the workcell is completed by establishing interfaces between machines, the store and the transport system, defining the flow of parts between them. This is done by synchronizing the suitable

```
{transformation-machine
    is-a :  transformation-resource machine
    states :  loaded unload-ready available-capacity
        ; place slots
        loaded :
        unload-ready :
        available-capacity :
    actions :  load process unload   ; transition slots
        load :  load-M
        process :  process-M
        unload :  unload-M
    ; predicted information and data collection
    pending-operations :
    statistics :
    max-part-size :   ; physical characteristics
    constraints :
    set-up :
    average-processing-time :
    precision-level :   ; abstraction hierarchy relations
    has-resources :
    resource-of :
    type-of-synchro :   ; interface features
    input-synchro :
    output-synchro :
    icon :   ; graphic representation characteristics
    wickname :
    comment :     }

{load-M
    is-a :  transition
    pre-net-relations : (& available-capacity AFtake)
    post-net-relations : (& loaded AFobj)
    associated-data : <obj> <op>
    predicate : (resource <op>) = &
    ao-conflict-of :   }

{unload-M
    is-a :  transition
    pre-net-relations : (& unload-ready AFobj)
    post-net-relations : (& available-capacity AFput)
    associated-data : <obj>
    predicate : t
    ao-conflict-of :   }

{process-M
    is-a :  transition
    pre-net-relations : (& loaded AFprod)
    post-net-relations : (& unload-ready AFnextop)
    associated-data : <obj> <op>
    predicate : t
    ao-conflict-of :   }

    AFobj : (name <obj>)
    AFtake : (name one)
    AFput : (name one)
    AFprod : (name <obj>; operation <op>)
    AFnextop : (name <obj>; operation (next <op>))
```

Figure 3: Composed object modelling the machine prototype.

actions (e.g., `load` and `unload` actions of machines are synchronized with the `pick-and-place` action of robot, the results are the `load-M`$_i$ transitions). Figure 5 shows the HLPN modelling the dynamic behavior of the complete workcell `C1`.

To complete the system model rests the establishment of the initial state. It is performed by means of the disposition of token objects in place slots. Figure 5 shows also, this initial state in a graphical way.

3 Dynamic aspects

This section deals with the underlying high level Petri net of a KRON model. Each transition has a set of variables which appear in expressions labeling the input and output arcs (defined by the net-relations). An expression defines possible bindings of variables with token attributes. The possible bindings of these variables are called *firing modes*. A transition is said that it is *enabled* when there are *tokens* in the input places for make the substitutions of the variables and these define the firing mode. If a variable appears in some input arcs, it must to get the same value in each of these. It is possible to constraint the set of the firing modes with a predicate which is defined on the set of variables of the transition. The change of the system state happens when an enabled transition is fired.

Following Petri net semantics, pre net-relations specify the token objects, that must be removed from the place slots when a transition object is fired with respect to a certain firing mode. The post net-relations, established from a transition object to a place slot, specify the token objects that must be saved in the output place slots for a particular firing mode of the transition object.

In the proposed example, the transitions `load-M`$_i$ and `unload-M`$_i$ model the activity for to load and to unload the machine i, whereas `process-M`$_i$ model the processing of an operation in the machines.

In the net of the figure 5, there are three enabled transitions: `load-M1` by the firing modes which are defined by the bindings {`<obj>=P1-01,<op>=OP1`} and {`<obj>=P1-02,<op>=OP1`} , `load-M3` by the same firing modes and `unload-M2` by {`<obj>=P2-01`} .

The firing, of an enabled transition, retires tokens from the input places and puts them in the output places. The firing of transition `load-M1` respect the firing mode {`<obj>=P1-01,<op>=OP1`} involves:

- The token object `P1-01` is removed from the place slot `stored-parts` of the `ST` state object.
- The token objects `NEUTRAL` (it is a token object without information) represented by $< \bullet >$ are removed from the place slots `available-capacity` of `M1` and from `free` of `R`.
- The token object `P1-01` is put in the place slot `loaded` of the `M1` state object.

This firing means that the machine `M1` has been loaded with piece `P1-01`.

However, a KRON model can be not fully deterministic, there exist points in which decisions have to be taken in order to establish the model evolution. From the underlying Petri net point of view, these decision points correspond to conflicts, that is, situations where there exist several excluding firing modes. In the example, all the enabled firing modes are in mutual exclusion. It must

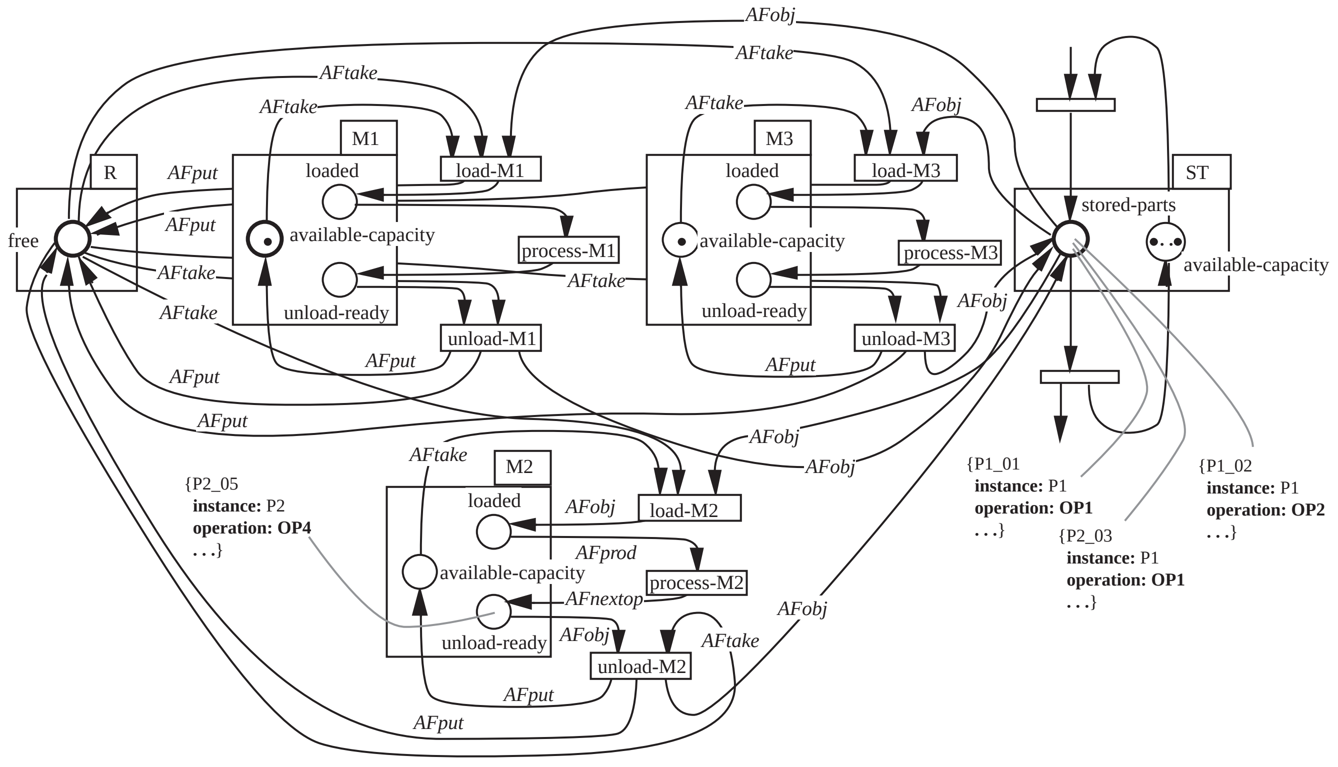

Figure 5: Structure of Petri net underlying in the model of workcell `C1` at the machine level of abstraction.

decide the operation that must perform the robot: to load machine `M1` or machine `M3` with part `P1-01` or part `P1-02`, or to unload the machine `M2`. KRON provides a mechanism to represent these decisions points: the *conflict objects*. A conflict object groups transitions which can pose a conflict and associates to it a resolution strategy called *control policy*. The *structural conflict* Petri net concept allows the automatically grouping of transitions. This technique enables us to establish a simple interface between the model and a decision making system.

4 Model behaviour interpretation mechanism

The underlying HLPN provides the behavioural rules for the model evolution. Based on these rules, KRON provides an efficient execution mechanism, which can be seen as a centralized implementation of HLPN [6]. The execution mechanism of KRON makes use of the similarities with the inference engine of a rule base system. A HLPN can be interpreted as a rule based system in which transitions have the role of rules and tokens are the working memory elements. These similarities have been pointed out in several papers [4], [8]. In these papers an efficient rule based language as OPS5 is proposed to implement HLPN.

However the HLPN provides some features that make its implementation more specific than in general rule based systems. The execution mechanism of KRON implements a strategy to specialize the RETE match algorithm to be more suitable for HLPN implementations [2].

As in RETE, the main objective is to exploit the data temporal redundancy. With this purpose, a RETE-like data structure is implemented. Additionally, our approach benefits from the partition of working memory provided by the HLPN that implies simpler data structures than the ones in more general production systems such as OPS5 [12].

Other difference from OPS5 implementations is the process of removing an element from the working memory. This process is not systematically repeated in a general rule based application. However each transition firing implies remove tokens. For this reason RETE is adapted to avoids the inefficiency due to unnecessary searches.

In other centralized implementations the main emphasis is paid to the selective scanning of transitions. The idea is to make a quick selection of a subset of transitions likely to be fired, avoiding the checking of all transitions. The consideration of these not enabled transitions makes the efficiency of the centralized interpreters to decrease.

The operation of the KRON interpreter is incremental, and comparing with other non incremental centralized implementations, is more efficient because avoids recalculations and considers only totally enabled transitions.

5 Conclusions

This work deals with the modelling of *discrete event systems*. High level Petri nets have been recognized as a suitable mean of to describe complex discrete event systems. However, high level Petri nets are still lacking of aspects such as methodological and data representation aspects.

This paper presents KRON (Knowledge Representation Oriented Nets). The main objectives in the definition of KRON have been: 1) to obtain an overall and powerful representation model for data and control; and 2) to incorporate a powerful modelling methodology. KRON is based on the integration of high level Petri nets with frames and an object based methodology. Frame related concepts have been used as a basis to support the representation due to their expression power to represent concepts and theirs relationships. On the other hand, the object based methodology supports conceptual models near to human conceptualization and independents from implementation, and facilitates reusability and model extensibility based on encapsulation and inheritance characteristics.

KRON supports a more high level mechanism to communicate objects than the classical message passing, which is based on transition synchronization. It allows to separate the model from the implementation and do not restrict the model to the client-server framework.

KRON provides an efficient execution mechanism. It has been designed to take advantages of data temporal redundancy. The implementation of the KRON interpretation mechanism is an adaptation of the RETE matching algorithm to deal with the features provide by HLPN. Moreover, KRON allows the representation of decision points in the execution of models. This mechanism enables us to establish a simple interface between the model and a decision making system.

KRON has been implemented in KEE. It constitutes the kernel of a modelling environment which is specialized in manufacturing systems. This environment is composed by others modules such as graphical interfaces, simulators or making decision systems.

Acknowledgments

This work has been supported in part by project ROB91-0949 from the Comisión Interministerial de Ciencia y Tecnología of Spain and project IT-10/91 from the Diputación General de Aragón. J. A. Banares is FPU fellow from the Ministerio de Educación y Ciencia of Spain.

References

[1] Baldassari, M., and Bruno, G. Protob: An object oriented methodology for developing discrete event dynamic systems. *Computer Languages 16*, 1 (1991), 39–63.

[2] Banares, J. A., Muro-Medrano, P., and Villarroel, J.L. Taking advantages of temporal redundancy in high level petri nets implementations. In *Proc. of 14th International Conference on Application and Theory of Petri Nets* (Chicago,Illinois,USA, June 1993), pp. 32–48.

[3] Battiston, E., Cindio, F. D., and Mauri, G. *Advances in Petri Nets 1988.* No. 340 in Lecture Notes in Computer Science. Springer Verlag, 1988, ch. OBJSA Nets: a class of high-level Petri nets having objects as domains, pp. 20–43.

[4] Bruno, G., and Elia, A. Operational specification of process control systems: Execution of prot nets using ops5. In *Proc. of IFIC'86, Dublin* (1986).

[5] Buchs, D., and Guelfi, N. Co-opn: a concurrent object oriented petri net approach. In *Proc. of the 12th International Conference on Application and Theory of Petri Nets* (Gjern (Denmark), June 1991), pp. 432–454.

[6] Colom, J., Silva, M., and Villarroel, J. On software implementation of petri nets and colored petri nets using high-level concurrent languages. In *Proc of 7th European Workshop on Application and Theory of Petri Nets* (Oxford, July 1986), pp. 207–241.

[7] Dileva, A., and Giolito, P. High-level petri nets for production system modelling. In *Proc. of the 8th European Workshop on Application and Theory of Petri Nets* (Zaragoza (Spain), June 1987), pp. 381–396.

[8] Duggan, J., and Browne, J. Espnet: expert-system-based simulator of petri nets. *IEEE Proceedings 135*, 4 (July 1988), 239–247.

[9] Giovani, R. D. Petri nets and software engineering: Hood nets. In *Proc. of the 11th International Conference on Application and Theory of Petri Nets* (Paris (France), June 1990), pp. 123–138.

[10] Jensen, K., and Rozenberg, G., Eds. *High-level Petri Nets.* Springer-Verlag, Berlin, 1991.

[11] Sibertin-Blanc, C. High-level petri nets with data structures. In *Proc. of Workshop on Applications and Theory of Petri Nets. Finland* (June 1985).

[12] Valette, R., and Bako, B. Software implementation of petri nets and compilation of rule-based systems. In *11th International Conference on Application and Theory of Petri Nets* (Paris, 1990).

[13] van Hee, K., and Verkoulen, P. Integration of a data model and high-level petri nets. In *Proc. of 12th International Conference on Application and Theory of Petri Nets* (Paris, 1991), pp. 410–431.

[14] Varaiya, P., and Kurzhanski, A. E. *Discrete Event Systems: Models and Applications.* Lecture Notes in Control and Information Sciences, 103. Springer-Verlag, 1988.

[15] Vautherin, J. *Advances in Petri Nets 1987.* No. 266 in Lecture Notes in Computer Science. Springer Verlag, 1987, ch. Parallel Systems Specifications with Coloured Petri Nets and Algebraic Specifications., pp. 293–308.