

A High Level Specification Mechanism for the Analysis and Design of Manufacturing Systems*

S. Ramaswamy**, K. S. Barber

*The Laboratory of Intelligent Processes and Systems, The Department of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712-1084*

ABSTRACT

In this paper, a structured development approach to deriving system models that helps to better understand and represent system details is presented. Real-world systems are never completely designed by either top-down or bottom up approaches. Therefore, methods that allow the integration of these two approaches as well as provide a means for analyzing such models are very useful. First, a general purpose system architecture, a corresponding system specification language, and a broad classification of the various concurrent system activities are presented. Second, Petri net (PN) extensions to simplify system specification and modeling are provided. Third, a transformation between the high level specifications to the proposed PN extensions is provided. Finally, a PN system model using the new PN extensions is derived as an example. To facilitate the use of existing PN based analysis and simulation tools a transformation between the PN extensions used in this paper to classical PNs is also illustrated.

I. Introduction

Most real-world systems are usually characterized by complex interactions between a variety of components that include distributed and parallel software, hardware and communication components. In addition, they increasingly have a large number of interfaces (both internal and external) that aid in sensing the operating environment and other subsystems. The number of interconnections between their corresponding components tends to be large and complex. Moreover, these components evolve over time, their logical and physical interconnections change, and the operational semantics of the system change accordingly, often leading to increased system complexity. Understanding the overall coordination and integration of these subsystems, and their conflicting requirements specifications, rather than details about specific subsystem components, seems to be a major barrier. Such subsystems also interact in a complex manner with respect to shared system resources, data and other information. Thus, in the context of present day systems, the following are to be noted:

- Systems development is incremental, that is, more complex systems are built incrementally from simpler modules. System requirements may not be totally specified / defined from initial conception. Thus, incremental modules are appended to existing systems / system models and only such modules are designed, tested and integrated.
- Due to rapid technological developments, modern day systems development is more often than not carried out by teams of personnel operating over large distances, and also probably by teams that cannot communicate effectively. Other factors that may affect complexity are geographic distribution of processors and databases, human skills, and unpredictability of system reactions to unexpected sequences of external events. A

visual and dynamic modeling tool, that also serves as a powerful specification / simulation tool, like a PN representation modified to handle system specification, would immensely help in understanding and integrating these diverse efforts. Such a representation will also serve as a flexible mechanism to represent and communicate ideas between the different phases of system design and development.

Thus, capturing the system behavior by means of a dynamic system model becomes an invaluable advantage. Typical applications areas include manufacturing, process control, aerospace, defense, transportation, communications, energy, utilities, medical, health, and commercial data processing. Moreover, there exist a variety of system specification and modeling tools which are essentially independent of one another. Although these tools are used for analyzing well defined structural and behavioral properties of a system design, they are based on different modeling paradigms which cannot be easily integrated. A high level specification language is used to derive the high level specifications of system structural and behavioral characteristics. The advantage of using a high level specification is that it is independent of the modeling tool / methodology used for system analysis. The motivation behind this approach is that once all high-level descriptions of the subsystems and their interfaces are defined, the modeling and analysis procedures involved in the design of the individual subsystem functionalities are relatively independent. Therefore, given such a specification mechanism, extensions to the basic model characteristics can be made uniformly such that an abstract system specification is independent of modeling and analysis procedures. This high level specification formalism provides flexibility in using different modeling tools / methods for system analysis and development. To illustrate, PNs are used for modeling a small production process, where the high level system specification is first transformed into a high level PN system model. Then, techniques to integrate incremental system details into the system model, defined in terms of rules for expansion and reduction of the basic model attributes (in this case, PN places and transitions), may be used for the generation of a correct, executable, low level system model. This model can then be used for analysis and design verification.

Section II provides the system structure and a system specification language. In Section III, PN extensions are discussed with appropriate justifications. Special emphasis is placed on the generation of low level system models using top-down decomposition techniques from generalized high-level specifications. Section IV presents a small example to illustrate the approach and Section V concludes the paper.

II. System Structure

A typical manufacturing system basically consists of a group of interacting subsystems that are (i) distributed and networked, (ii) subsystem operations are either sequential or parallel, loosely or tightly coupled, synchronous or asynchronous in nature, with or without a global clock, (iii) error identification is localized; however error recovery may involve the cooperation of two or more

* This research was supported in part by the Texas Higher Education Coordinating Board under grants ATP-115 and ATPD-112.

** Dr. S. Ramaswamy is currently with the Division of Computer and Applied Sciences, Georgia Southwestern College, Americus, GA.

subsystems or independent processes. In this section, a system specification language is presented that simplifies the transition between high level system specification and the generation of the corresponding system model. Once the high level system model is generated, an executable model may be easily generated by introducing model-specific decomposition rules. This executable model can then be used in the analysis and validation of system operations. The incorporation of time into modeling tools such as PNs, StateCharts, or timed automata, simplify the analysis of response times, performance, and reliability. The advantages of this approach are: (i) It allows the defi-

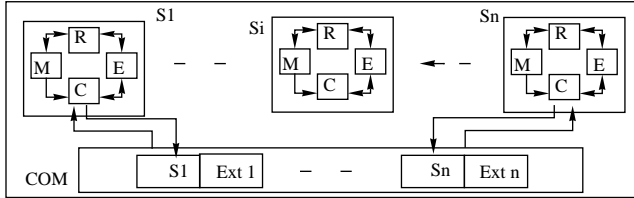


FIGURE 1. SYSTEM ARCHITECTURE

inition and realization of general plans, (ii) It simplifies the process of reasoning about the software, the algorithms used for scheduling, etc., since it provides a direct translation to an operational model. By enabling the direct correlation between design specification and system operations, the designer can easily modify and reason about modifications to the original specification and investigate related benefits, (iii) The verification of system properties and the evaluation of performance attributes is highly simplified, and, (iv) It provides a structured approach to the analysis and design of distributed systems. The definition of such a system architecture involves many steps, these include: (i) The definition of subsystems and respective operations, (ii) Determining system actions to perform a certain operation, and, (iii) Determining appropriate deadlines for macro level operations. Operations may include either monitoring or service functions. Service functions may be either synchronous or asynchronous. Monitoring functions are either periodic or event driven and can initiate local or global failure recovery. Such a system structure is shown in Figure 1. At the highest level of abstraction, three different kinds of basic activities are performed by each subsystem, namely, monitoring (M), processing (E) and communication (C). R in Figure 1 represents the set of resources associated with the subsystem. The communication module is further classified as either internal communication (between the various subsystems, s_1, \dots, s_n) or external communication (to other external interfaces, either software or human, ext_1, \dots, ext_n). The three basic activities are carried out simultaneously at any given instant during the system operation. However, the sub-activities associated with the three basic activities are not tightly coupled.

The high level system specification is by means of the system specification language shown in Figure 2. The specification language is a high level construct that is used to specify system operations, communication, synchronization and timing information. It is powerful and allows for the specification of groupings and sub-groupings of activities and also allows for the specification of fine-grain parallelism between the various sub-activities. “num” and “id” represent the set of numeric values and identifiers that stand for time values and the various labels in the system model. Similar to extended BNF syntax, | is used to represent alternatives, < > represents non-terminal symbols in the language, and, [] is used to represent optional syntax. In addition, similar to CSP notation, || is used to represent parallel activities and *guard_expr*: *exec_expr* is used to represent the evaluation of a guard function, *guard_expr*, before the execution of

an expression, *exec_expr*. That is, the expression *exec_expr* is executed only if the guard holds true.

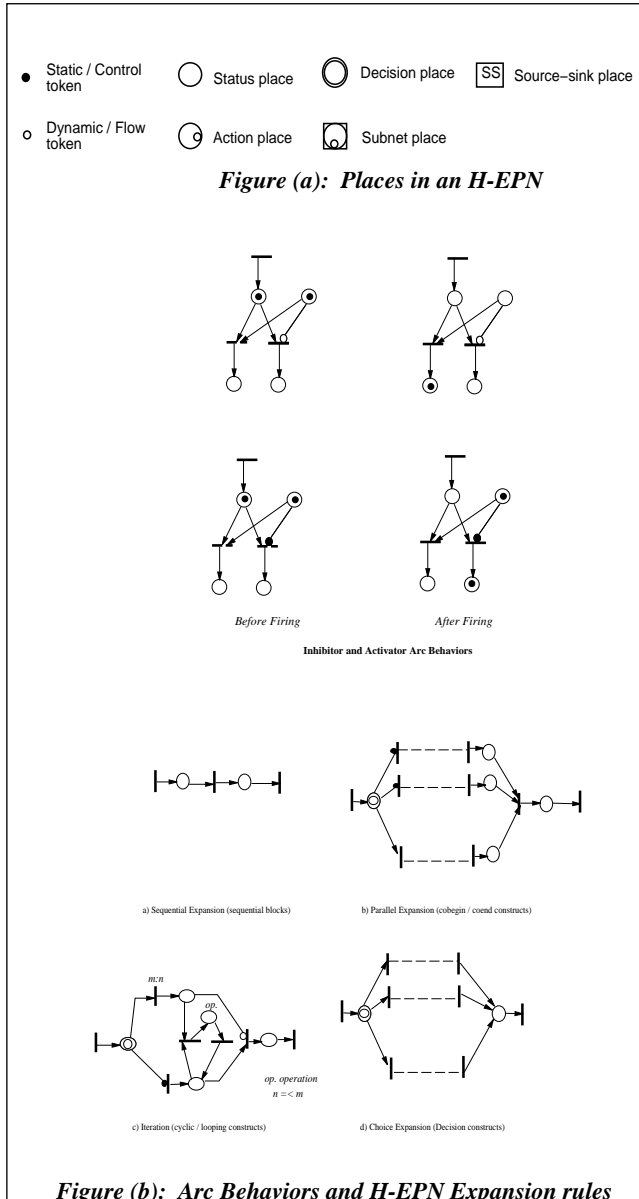
While the language in Figure 2 can itself be recursively used to derive low level system specifications, once the <oper_desc> have been identified for different sub-systems the specification trace obtained can be mapped to a high level modeling abstraction. In Figure 2, some labels are indicated in bold and others are italicized. Labels in bold and italic indicate the keywords of the specification language. Italicized words indicate operations that may not be a part of the model, but represent a specification of some external service (either from an external interface or from another independent sub-system). However, it is to be noted that periodic events are part of the system, but they can trigger operations that are not part of the subsystem description and hence they are italicized in the specification language.

<MAN-SYSTEM>:	<SUB-SYSTEM> <MAN-SYSTEM> <SUB-SYSTEM>
<SUB-SYSTEM> :	<SYSID><SYSREP> <SUB-SYSTEM><SUB-SYSTEM>
<SYSID> :	id
<SYSREP> :	<SUB-SYSTEM> <NODES>
<NODES> :	<NODE> [, <NODES>]
<NODE> :	<EVENTS><INPUT-NODES><NODEID> <OPER-DESC><OUTPUT-NODES>
<NODEID> :	<SYSID>.id
<INPUT-NODES>:	<NODES> [, <NODES>]
<OUTPUT-NODES>:	<NODES> [, <NODES>]
<EVENTS> :	<EVENT> [, <EVENTS>]
<EVENT> :	<i>input</i> <opeids> <i>output</i> <opeids> <i>timed</i> <eids> <i>error</i> <opeids> <i>periodic</i> <eids>
<OPER-DESC> :	<OPER-DESC> ; <OPER-DESC> <OPER-DESC> <OPER-DESC> <INTR-OPER> <OPER> <i>periodic</i>
<OPER> :	<EXEC-STMT> <SEND-STMT> <RECV-STMT> <LOOP-STMT>
<EXEC-STMT> :	[<i>guard-exec</i> (<opeids>):] [TIME] <i>exec</i> ()
<SEND-STMT> :	[<i>guard-exec</i> (<opeids>):] [PERIOD] <i>send</i> (<LIST>)
<RECV-STMT> :	[<i>guard-exec</i> (<opeids>):] [PERIOD] <i>recv</i> (<LIST>)
<LOOP-STMT> :	[<i>guard-exec</i> (<opeids>):] <i>loop</i> [TIME] <i>do</i> <OPER> <i>od</i>
<INTR-OPER> :	<i>do</i> <OPER> [<i>interrupts</i>] <i>od</i>
periodic :	<i>do</i> [PERIOD] <OPER> <i>od</i>
interrupts :	<i>timeout</i> [TIME] → <OPER> <i>case error</i> <opeids> → <OPER>
<LIST> :	<NODEID> [, <NODEID>]
<eids> :	eid [, <eids>]
<opeids> :	eid [<op> <opeids>]
<op> :	operator
eid :	id
TIME :	num
PERIOD :	num
watch-dog :	<i>if timed</i> <eids> : <i>query</i> (LIST); <i>do</i> <INTR-OPER> <i>od</i>

FIGURE 2. THE SYSTEM SPECIFICATION LANGUAGE

Timing information in the system specification is discretized. TIME indicates a hard deadline while PERIOD indicates a soft deadline. That is, if a process specified with a TIME value, τ , does not complete in τ time units after it begins execution of some operation an error event is generated to indicate some system error. This causes the system to initiate error handling with respect to timing lapses. The watchdog function can be attached to the input set of any node. This function monitors the status of execution of all or some of the nodes in the input set. A status query can be generated by the watch-

dog function using the reserved word query. Depending on the status information the watchdog function initiates necessary error handling. The kind of error handling initiated may be as simple as resetting all or part of the operations associated with the particular node or as complex as generating hard system failures.



III. Petri Net Extensions

The basic PN structure is of limited use in the specification and modeling of time-dependent systems and it lacks in both functional and timing specifications. Extensions that provide additional specification capabilities are of immense theoretical interest while those that provide modeling flexibility by means of functional extensions are of immense practical value. For a system modeling tool to be complete for the modeling of distributed time-dependent systems the issues that need to be addressed are: (i) The modeling tool must allow the use of net structuring mechanisms that can be specified with nested levels of resolution, (ii) The tool must allow for the formal analysis of system properties and it must be possible to extract information about the system performance, and, (iii) The model must

allow the building of looping structures where loop variables can be time bounded and it must also allow some means to specify a bounded choice for loop iterations. It is assumed that the reader is familiar with the basic PN structure. For details on Hierarchical Time-Extended Petri Nets (H-EPNs) the reader is referred to [4].

In this research, we specify PN extensions with respect to Hierarchical Time-Extended PN, or H-EPNs. Although different kinds of H-EPN places have been defined, it can be noticed that these extensions have been used only to increase the expressive power of the basic PN structure. Thus, given the above, a H-EPN structure can be easily transformed to a basic PN structure for analysis and verification. An H-EPN system model is described by a 4-tuple: $H-EPN = (P, T, A, M_0)$ where, P is a set of places (see Figure 3a) and is further classified as P_s, P_a, P_d, P_{su} and P_{ss} , T is a set of transitions that are further divided into external and internal transitions depending on the events (internal events are known apriori and modeled, external events are known to require human intervention, and are not modeled) associates with the transitions, A is a set of arcs in the H-EPN model. Arcs are of three different kinds. In addition to normal and inhibitor arcs as in ordinary PN extensions, H-EPNs have an arc extension called activator arcs that are similar in structure to inhibitor arcs but have a contrasting functional specification. Thus:

$$A = A_N \cup A_I \cup A_A, A_N \subseteq P \times T \cup T \times P, A_I \subseteq P \times T, \text{ and } A_A \subseteq P \times T$$

A_I, A_A denote inhibitor and activator arcs respectively and A_N represents other arcs that are similar to arcs in a classical PN model, M_0 is the initial H-EPN marking. The activator arc extension has been introduced in simplifying system specification characteristics as well as providing additional flexibility in functional specification. Although activator arcs have been introduced as a means to add more expressive and specification power to the basic PN model, PNs with activator arcs are easily transformed into PNs without activator arcs. This transformation makes it very useful for the verification of PN structural and behavioral properties using already available PN simulation packages. The activator arc extension is similar to the inhibitor arc extension in that tokens do not flow along these arcs and that they are arcs drawn only from places to transitions in the system model. However, the behavior of activator arcs is contradictory to that of inhibitor arcs. An activator arc from a place p_i to a transition t_j implies that if p_i is marked, it enables the firing of t_j , while an inhibitor arc from a place p_i to a transition t_j implies that if p_i is marked the firing of t_j is disabled. The behavior of activator and inhibitor arcs is illustrated in Figure 3b. Activator arcs have also been useful to design, analyze and integrate PN subnets that have a multiple-input multiple output (MIMO) structure [5]. In [1] and [6], arcs similar in function to activator arcs are defined and used in the specification of real-time systems. However, they do not implicitly define transition priorities. The advantages gained by using activator arcs include [5]: (i) Simplification of subnet construction and analysis, (ii) visualization of priorities, (iii) easier communications modeling. The definition of expansion and reduction rules for H-EPNs allow the integration of the above factors in a H-EPN model. These expansion rules are illustrated in Figure 3b.

PNs with activator arcs can be easily transformed into PNs without activator arcs, thereby allowing the usage of existing PN simulation and analysis tools. Since most PN simulation tools allow for discrete time PNs, H-EPNs are easily analyzed using such tools. Figure 4a and Figure 4b illustrate the transformation algorithms between PNs with activator arcs to PNs without activator arcs (Figure 4a) and between PNs with activator arcs to PNs with transition priorities but without activator arcs (Figure 4b). Figure 4c illustrates the two algo-

rithms with a simple example.

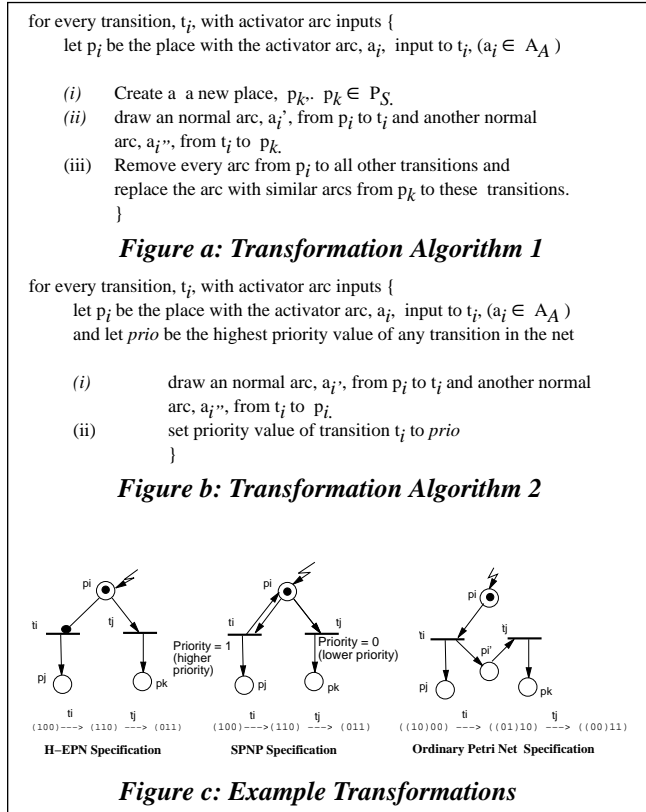


FIGURE 4. TRANSFORMATION ALGORITHMS

IV. Example: An Automated Assembly Process

In this section, the methodology introduced in the previous sections is used to generate an executable H-EPN model of an automated assembly processing system [7], shown in Figure 5. Note that the complete model has not been derived here. The system is composed of three workstations W1, W2, and W3, which process incom-

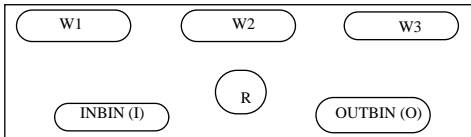


FIGURE 5. EXAMPLE

ing parts sequentially. A robotic system, R, is used to transfer parts between workstations that do not have the capability to load/unload parts. The robot can also be used in the processing stage of the workstations to hold / move partly processed parts. While W1 and W3 can operate on only one part at any given time, W2 can operate on two parts simultaneously. Moreover, W2 can also load/unload itself without using the robot. Thus R is shared between W1 and W3. The derivation of the four-layer H-EPN structure for a H-EPN system model of the above example at the coordination level has already been derived in [4]. Inputs arrive at the INBIN and are processed by the workstations sequentially and are removed from the system through the OUTBIN. The control software associated with the various subsystems fall into four main processes, namely, input, execution, monitoring and output processing as shown in Figure 6. It can be noticed that the specification is quite easy and highly generalized so that detailed system operations are easily abstracted in the specification. However, once the system model is generated additional details can

be easily added, modified and analyzed. Figure 7 is a trace the high

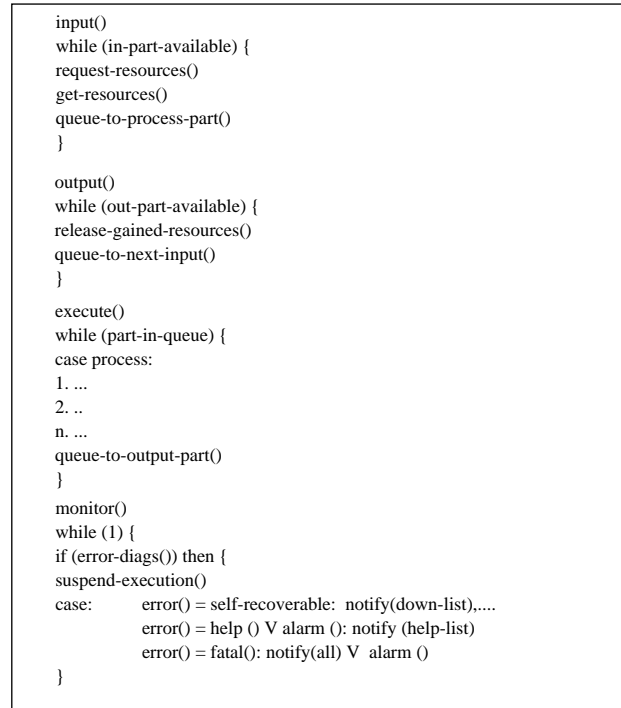


FIGURE 6. SYSTEM PROCESSES

level system specification. The equivalent PN model of the above processes is represented by a H-EPN model as shown in Figure 8. It

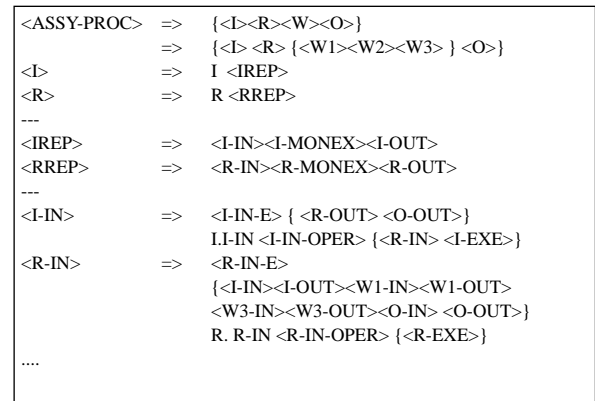


FIGURE 7. SPECIFICATION TRACE

has to be noted that the model is just a high level representation of all the processes and the entire system model is made of all such individual models. The MIMO-SISO transformation technique is used to abstract similar subsystem operations in the overall model, thereby giving a complex structure to the H-EPN system model.

Figure 9 gives the overall H-EPN system model. It can be observed that the system model is made up of independent modules that resemble the PN model in Figure 8. Subnets x and y can be further expanded to add additional details. Subnets $x1$, $x2$ and $x3$ are expanded by parallel expansion to represent information regarding different kinds of operations that each workstation is capable of performing on the incoming part. Subnets y in different subsystem areas are expanded by means of parallel expansions to add specific, localized monitoring activities. Moreover, this model can be easily transformed to the model in [4] which gives the four-layer coordination

level system model structure when related operations are combined as single subnet place in the H-EPN system model, where the subnet is context sensitive. That is, the semantics of the operations performed are dependent on the higher-level net that initiates the subnet operations. It is to be noted that in [4] only certain errors are modeled and therefore there do not exist a separate group of places and transitions that model the monitors in each subsystem. Moreover, the net in [4] provides a higher level abstraction of the system such that execution processes in W1, W2 and W3 are represented as a single context sensitive subnet place. Also explicitly modeled in [4] is the robot move operation and associated errors. This has not been derived here because the exercise in [4] was to derive the 4-layer coordination level structure, whereas the emphasis here is to derive a model that captures various concurrent processes in each subsystem from the system specification. Using the 4-layer structure derived in [4], an efficient grouping of system operations is then derived to aid complete system analysis.

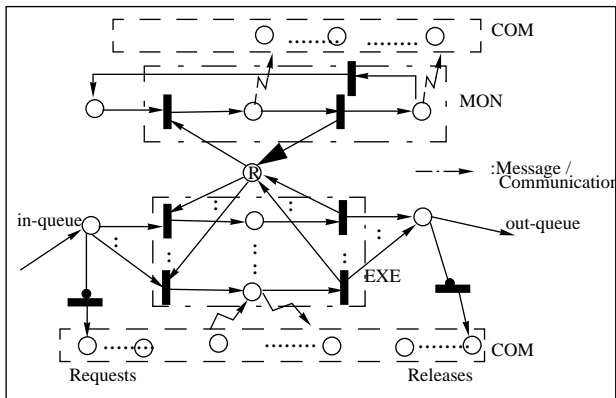


FIGURE 8. SINGLE H-EPN MODULE

V. Conclusions

A simple high level textual specification approach has been introduced. Incremental system details are easily added to high level system models that are generated from this high level specification. This

systematic approach leads to well defined low level system models of complex systems that lend themselves easily to changes, modifications and analysis. This research and the one reported in [2], [3] are directed toward generating implementation independent models of complex systems, including automated manufacturing systems.

References

1. Bandinelli, S. C., Fuggetta, A., Ghezzi, C., "Software Process Model Evolution in the SPADE Environment", *IEEE Transactions on Software Engineering*, Vol. 19, No. 12, December 1993.
2. Barcio, B. T., Ramaswamy, S., Barber, K. S., "An Object-Oriented Modeling and Simulation Environment for Distributed, Reactive Systems", *submitted to IEEE Transactions on Software Engineering*, Feb. 1995.
3. Barcio, B. T., Ramaswamy, S., Barber, K. S., "Object-Oriented Analysis, Modeling and Simulation of a Notional Air Defense System", *IEEE Internl. Conf. on Systems, Man and Cybernetics*, Oct. 1995.
4. Ramaswamy, S., Valavanis, K. P., "Hierarchical Time-Extended Petri Nets based Error Identification and Recovery in Multi-level Systems", *to appear in the IEEE Transactions on Systems Man and Cybernetics*, Feb. 1996.
5. Ramaswamy, S., Valavanis, K. P., Barber, K. S., "On the Construction and Analysis of Multiple-Input Multiple-Output Subnets", *to be submitted to the Journal of Manufacturing Systems*, June 1995, *earlier version appeared in the Proceedings of the International Conference on Automation, Robotics and Computer Vision, Singapore*, Nov. 1994.
6. Ritcher, G., Maffeo, B. "Towards a Rigorous Interpretation of ESML - Extended Systems Modeling Language" *IEEE Transactions on Software Engineering*, Vol. 19, No. 2, 1993, pp. 165-180.
7. Zhou, M. C., DiCesare, F., "Petri Net Synthesis for Discrete Event Control of Manufacturing Systems", Kluwer Academic Publishers, Boston, 1993.

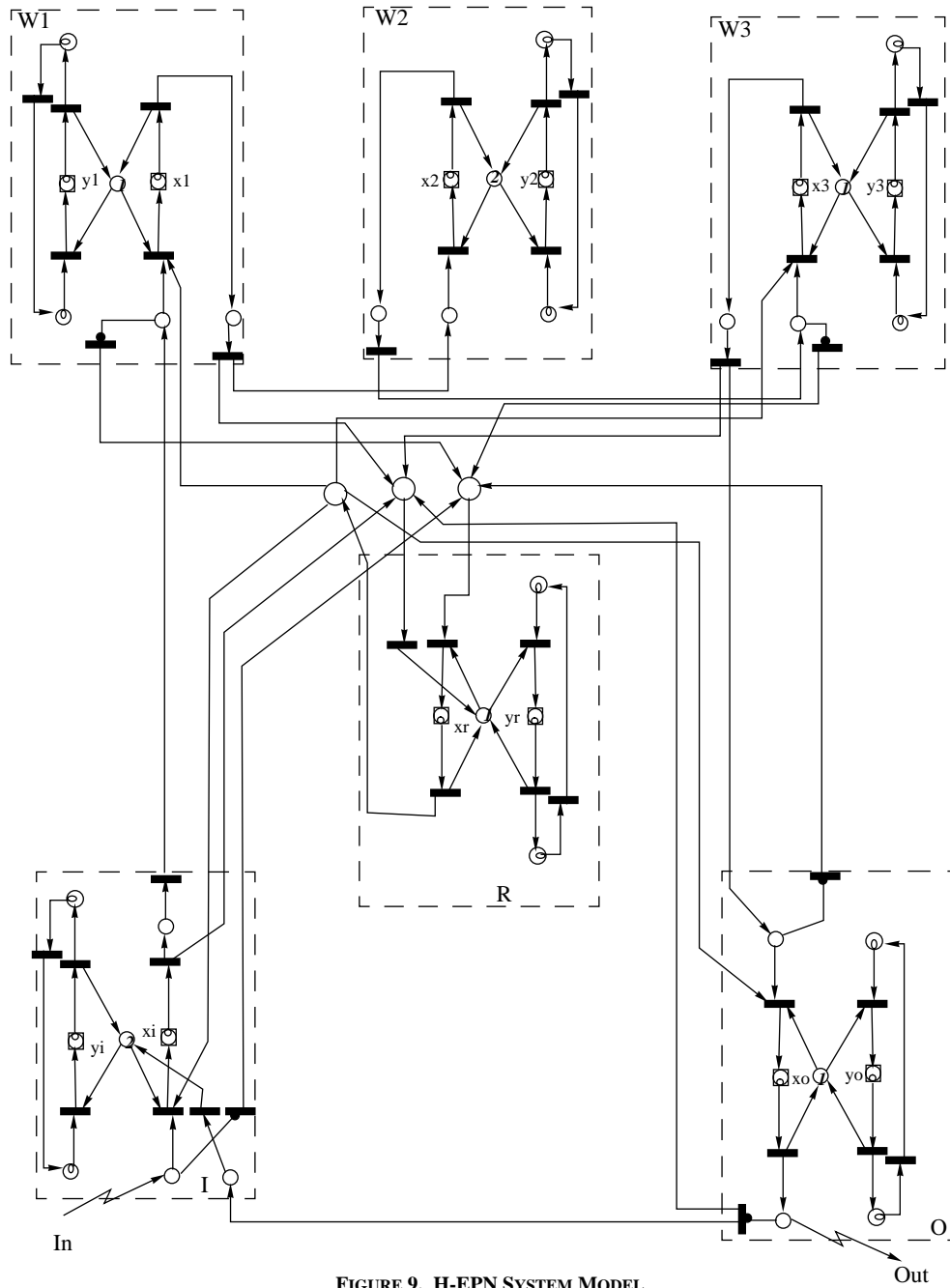


FIGURE 9. H-EPN SYSTEM MODEL