MAY 1989 LIDS - R - 1868

EVALUATION OF FUNCTIONALITY IN DISTRIBUTED SYSTEMS

by

FRANCOIS RENE HENRI VALRAUD

Ingénieur de l'Ecole National Supérieure de Mécanique
(1986)

This report is based on the revised and edited thesis of François Valraud, submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master of Science in Technology and Policy at the Massachusetts Institute of Technology in March 1989. The research was conducted at the Laboratory for Information and Decision Systems, with support provided by the Office of Naval Research under Contract No.N00014-K-85-0782.

Laboratory or Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139

EVALUATION OF FUNCTIONALITY IN DISTRIBUTED SYSTEMS

by

FRANCOIS R. H. VALRAUD

ABSTRACT

A new quantitative methodology is presented for identifying and evaluating the shortfalls and the overlaps between the desired functionality of a distributed decision making system, as characterized by the design requirements, and the functionality of a proposed or implemented distributed system. First, compatible Petri Net models of both the requirements and the system are presented. Second, a correspondence is established between structural properties of the Petri Net representation of the system and the functions it performs. On the performance side, a functionality is defined as a set of coordinated functions that a system must be able to carry out in order to accomplish a task. On the structural side, simple and complete information flow paths, i.e., sub-nets of a Petri Net, are defined. Thus a particular functionality is identified as a sub-net of the Petri Net representation of the system. Algorithms are presented to determine the various simple and complete information flow paths in the nets representing both the system and the requirements. This is accomplished through extensive use of the invariant theory of Petri Nets. The next step is the comparison of the two sets of flow paths to determine shortfall and overlaps. The formulation of the problem requires a set of definition that characterize the shortfalls and the overlaps in a precise manner. The methodology is illustrated by applying it to a hypothetical command center.

Thesis Supervisor	:	Dr. Alexander H. Levis
Title	:	Senior Research Scientist

ACKNOWLEDGEMENTS

I wish to express my gratitude to:

Alexander Levis for introducing me to the world of research. His high requirements, his professional excellence, his willingness to make such an experience an integrated part of one's education made this research, from its start to its completion a valuable experience. His guidance and support have been very important to me throughout this endeavour.

Jacques Demaël for his moral support, and the endless discussions we had over some technical issues of the Petri Net theory.

Didier Perdu, a veteran of this research group, for his moral support, and his help in the software implementation of some algorithms.

Victoria Jin, for her kindness, and her ability to bear with me in the office.

Stamos Andreadakis, for his support and guidance, his technical excellence, and the arguments we had about the best way to make coffee.

Jinane Abounadi for her extreme patience with me.

Sabina Skulsky for teaching me what a healthy life should consist of.

This research was conducted at the MIT Laboratory for Information and Decision Systems with support provided by the Basic Research Group of the Joint Directors of Laboratories through the Office of Naval Research under Contract no. N00014-85-K-0782.

To Antoinette, Catherine, Marie-Jeanne, and Jean

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 PROBLEM DEFINITION

There are two distinct approaches to the design of distributed systems. In the first one, given some set of requirements and constraints, an algorithm is used to obtain all architectures that meet these requirements and satisfy these constraints (Remy and Levis, 1988; Andreadakis and Levis, 1987). Then the different designs are evaluated, and the best performing system is selected. While this procedure is rigorous and guarantees that requirements will be met, it can only handle restricted classes of systems.

The second, more common approach, is for a group of designers to construct a large distributed system by interconnecting known subsystems or components. However, there is no guarantee that the proposed design will satisfy all its requirements and constraints. A trial and error procedure is used in which the system is tested and modifications are made when requirements are not met. The validation of such large scale distributed system is generally made through the use of testbeds and simulations. Testbeds cannot be built for testing all types of possible scenarios, and they require the allocation of extensive resources. Simulations, although they require more limited resources (and are often performed at an early stage of the design process,) generally fall short when they run into combinatorial explosion. Thus, despite its increasing importance, the assessment of the functionality of a distributed system is hard to accomplish.

Thus, a methodology is needed for modeling distributed systems, for developing a compatible representation of the requirements for a system and of its proposed or implemented counterpart, and for comparing the two representations.

Part of the modeling of distributed systems in this work is based on the Modular Command and Control Evaluation Structure (MCES) (Sweet 1986), a top-down procedure that guides the analysis and evaluation of the effectiveness of command and control systems. It consists of seven modules which, when followed, lead to a complete evaluation. The MCES can be perceived, at one level, as a simple checklist that identifies the major steps of the evaluation; at another level, it can be construed as a methodology. One of the strengths of the MCES is that it can accommodate different models and quantitative techniques in any specific instantiation of the modules. It can also accept a wide variety of data inputs: from simulations, testbeds, and war games. In previous applications of the MCES, emphasis was placed in the first three modules; very few studies carried out the analysis through the seven modules. In one recent study, the problem of concern was the evaluation of the architecture of a survivable, enduring command center (SECC). A modular and austere command center was being developed for the Strategic Air Command and was being prepared for the test and evaluation phase. The question that arose, given the complexity of the system called Proof of Concept/Experimental Testbed (POC/ET), is whether MCES could be applied to determine potential shortfalls and overlaps in the functionality of the system prior to the actual Test and Evaluation procedure. A study was carried out in which the first three modules of MCES were applied to the problem, and then a procedure was outlined for carrying out the following two. The purpose of this study is to develop the described procedure and test it out by applying it to POC/ET.

The purpose of this work is to develop quantitative procedures for implementing the fourth module of the methodology. This fourth module, Integration (of Elements and Functions), is the one in which the dynamic model of the system is obtained. This model is the one that generates the data necessary for the evaluation of the measures. The specification of the Measures is done in the fifth module, while the Data needed are generated in the sixth one. Finally, the measures are aggregated in the seventh module so that the answer to the evaluation problem be obtained. The fifth, sixth, and seventh modules can be implemented with quantitative procedures using the System Effectiveness Analysis (SEA) approach. The latter leads to the specification of a class of Measures of Effectiveness (MOEs) that is based on a set of application-specific measures of performance. The approach that is developed for the fourth MCES module is appropriate for the evaluation of shortfalls and overlaps - with respect to the requirements - in the architecture of command and control systems.

1.2 BACKGROUND

A distributed system consists of both humans and equipment such as computers, software, and communication devices. These components are assembled in a number of units spread out geographically for various reasons. Each unit performs a part of the processing necessary for the achievement of the goals of the system. These goals may be numerous, and the way to achieve them may be different depending on the circumstances in the direct environment of the system. Communication is needed to transmit information about the environment, to send information from one unit to another, and to transmit the response(s) of the system to the devices that act upon the environment, also called *actuators* or *effectors*. Indeed, a system is an integrated assembly of heterogeneous elements that are needed to perform a certain task or mission. A system is self sufficient in the sense that it does not need other components to perform its task.

The need for a distributed system arises often in cases such as:

- Technological limitations or limitations due to physical laws require the geographical dispersion of a system. This is the case for any Early Warning Control system, monitoring either aircrafts or battleships. The speed and range of the planes, and the range of radars (determined both by physics and technology) constrain the system to be physically distributed so that the whole air space can be monitored and controlled according to safety requirements.
- Timeliness requirements lead to the decomposition of the task into sub-tasks that can be performed concurrently so that the overall processing time is reduced in a satisfactory manner.
- Multi-user systems such as banking systems which provide, in different locations, electronic services for the more usual banking operations to their customers.

A distributed intelligence system is seen as an entity performing a mission. The processing of the mission is achieved through the execution of well defined procedures or algorithms that human decisionmakers, intelligent nodes, or supporting information systems have.

The mathematical formulation of the structure of the system is based on the theory of ordinary Petri Nets (Reisig, 1985). Petri Nets have been introduced to model organizational forms as they show explicitly the interactive structure and the sequence of operations between the components of the organization. Petri Nets have proven their efficiency as modeling and analytical tools.

Therefore, the approach taken in this work is based on comparing the functionality of the command center, as specified by the requirements, and of the proposed or implemented system architecture. Petri Nets are used to develop a graph representation of the required functionality. Analysis of this net using algorithmic and simulation tools establishes the required system functionality in terms of structural properties such as net invariants (e.g., the S-invariants of the Petri Net). Another Petri Net can be drawn, usually quite large, that represents the command center, either on the basis of the specifications and the design documentation or the actual system as modeled from its technical description and operational data. These two Petri Nets form the basis for the quantitative comparison of the desired and actual functionality. The concepts of shortfall and of overlap are analyzed in the context of the structural properties of these nets.

1.3 GOALS AND CONTRIBUTION

The goal of this work is to provide a methodology for assessing the functionality of distributed systems on the basis of their structural properties. A functionality is defined as a set of coordinated processes, or functions that a system must be capable of carrying in order to accomplish a task or subtask.

This report proposes a quantitative approach for the evaluation of proposed designs. The fourth module of MCES, the Integration of Elements and Functions, when applied to the problem of evaluating the design of a command center, can be divided into three steps, namely,

1. Determination of functional interrelationship between elements and processes;
2. Modeling of information flows; and
3. Determination of shortfalls and overlaps.

The assumption underlying the procedure to be described is that a set of requirements has been developed that any proposed design or architecture of the command center must satisfy. These requirements may have been obtained using the Mission Oriented Approach (MOA) (Signori and Starr, 1987), or any other less general methodology that is appropriate for the case at hand. Then, one or more scenaria need to be chosen that would excite the various operating modes of the system. The requirements, when interpreted in the context of a scenario, lead to the construction of a Petri Net that depicts the interrelationship of the various functions. Different Petri Nets can be obtained for different scenaria, because not all functions are active and in the same sequence for each scenario. For the requirements net, only analysis of the interrelationship between processes needs to be done. This is accomplished by determining the S-Invariants of the net and from those constructing the simple and the complete information flow paths. Those, in turn, are interpreted as representing all the simple and complete functionalities that are implicit in the requirements. A functional evaluation should permit the designer to identify shortfalls and overlaps between the requirements and the implemented or proposed design. A shortfall is defined as a functionality which is required but not present in the actual implemented or proposed design. An overlap is defined as an unnecessary redundance of functionality. The notion of overlap is particularly difficult to evaluate; necessary redundancy for reliability or survivability should be distinguished from unnecessary redundancy that may cause congestion and delays. Lastly, the issue of coordination of functionalities is addressed.

1.4 OUTLINE OF THE REPORT

Chapter II provides an introduction to Petri Net theory. The mathematical framework of Petri Nets is used in this work to model distributed systems and to provide tools for the analysis of the proposed or implemented system and of the requirements.

Chapter III presents a procedure for modeling the requirements and the actual distributed system. The resulting models do emphasize certain aspects of distributed systems that are relevant for the analysis of functionality. The concept of functionality is defined, and some sub-nets are identified that correspond to the system's functionality.

Chapter IV presents the procedure for the evaluation of the functionality of a distributed system. With the establishment of relations between the functionality and some sub-nets, the basis of the evaluation is the comparison of the sub-nets embedded in the two Petri net models: that of the requirements and that of the proposed or implemented system. Several definitions are provided that describe different types of shortfalls and overlaps.

Chapter V presents a generic mobile, endurable, survivable, and austere, command center, whose mission is air interdiction, to illustrate the two types of nets that are necessary for the quantitative analysis and the evaluation procedure of such a defense system. The example is based on POC/ET (Proof of Concept/Experimental Testbed) which is an implementation of the Survivable Enduring Command Center (SECC) concept that is being tested by the Strategic Air Command.

In Chapter VI, the results obtained are summarized, and some directions for further research are suggested.

CHAPTER II

PETRI NET THEORY

Petri Nets are a very convenient tool for modeling and analyzing concurrent and asynchronous processes. Since information processing in distributed systems exhibits such properties, Petri Nets have been used for their modeling (Tabak and Levis, 1985). Petri Nets show explicitly the structure of the interactions between different processing units within a system, and allow their study at different levels of aggregation. This chapter reviews the basic definitions of Petri Nets; more introductory material can be found in Peterson (1981), Brams (1983), and Reisig (1985).

The Petri Net formalism is a tool for modeling or describing a discrete event system and then analyze it. Analysis is used to obtain insights into the interrelationships between the processes modeled and to understand fully what the representation implies. Analysis based on invariant methods relates the structural properties of a Petri Net with behavioral properties of the system it represents. Invariant methods, when feasible, are often more practical than reachability methods or simulations which often lead to combinatorial explosion. This chapter provides an introduction to some invariant methods of analysis.

2.1 PETRI NETS

2.1.1 Basic Definitions

Definition 2.1:

A *Petri Net* is a bipartite directed graph represented by PN = (P,T,I,O), where:

- P = {p1, p2,..., pn} is a finite set of places.
- T = {t1, t2,..., tm} is a finite set of transitions.
- I is a mapping from P × T to N, where N is the set of non-negative integers, corresponding to the set of directed arcs from places to transitions. I (p,t) = 1 means that the place p is connected to the transition t, in the sense that there exists a directed arc from p to t.
- O is a mapping from P × T to N, corresponding to the set of directed arcs from

transitions to places. O (t,p) = 1 means that there exists a directed arc from t to p.

Definition 2.2:

Given a node, x, its *postset*, noted x•, is the set of output nodes of that node. Similarly, •x is the *preset* of x, and represents the set of input nodes of x.

Therefore, we have:

$t\bullet = \{ p \in P \text{ such that } O(t, p) = 1\}$ and $\bullet t = \{ p \in P \text{ such that } I(p, t) = 1\}$

$p\bullet = \{ t \in T \text{ such that } I(p, t) = 1\}$ and $\bullet p = \{ t \in T \text{ such that } O(t, p) = 1\}$

Definition 2.3:

A Petri Net is said to be *ordinary* if and only if the mappings I and O take their values in the set {0, 1}.

In the sequel, all Petri Nets considered will be ordinary Petri Net.

Definition 2.4:

A Petri Net is said to be *pure* if and only if it has no self loop, i.e., no place can be both an input and an output of the same transition.

An example of Petri Net is shown in Figure 2.1. A place is depicted by a circle, a transition by a filled bar, and an arc by a directed line segment.

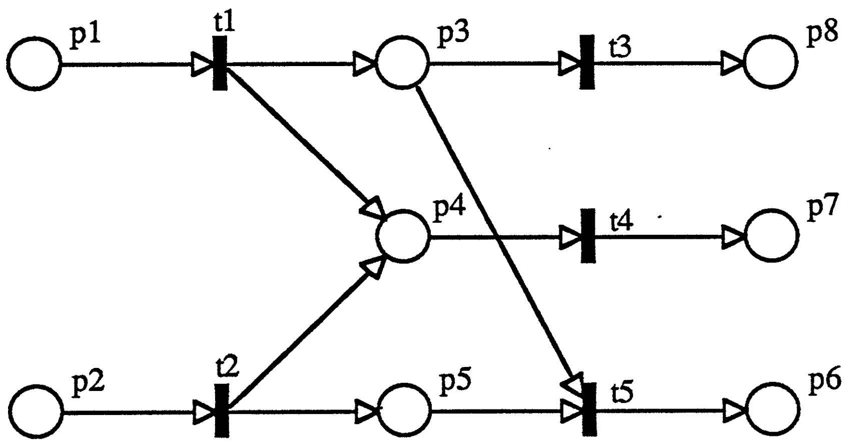


Figure 2.1 Petri Net PN1

A marking of a Petri Net is a mapping from P to N which assigns a non-negative number of tokens to each place of the net. This mapping is represented by a vector of non-negative integers. This vector has the dimension of the number of places, n. The number and positions of tokens may change during the execution of a Petri Net. On a Petri Net graph, tokens are represented by small dots, •, in the circles which represent the places of a Petri Net. Since the number of tokens which may be assigned to a place of a Petri Net is unbounded, there is an infinity of markings of a Petri Net. Figure 2.2. depicts Petri Net PN1 with the following marking:

M(p1) = 1	M(p2) = 0	M(p3) = 2	M(p4) = 1
M(p5) = 1	M(p6) = 0	M(p7) = 0	M(p8) = 0

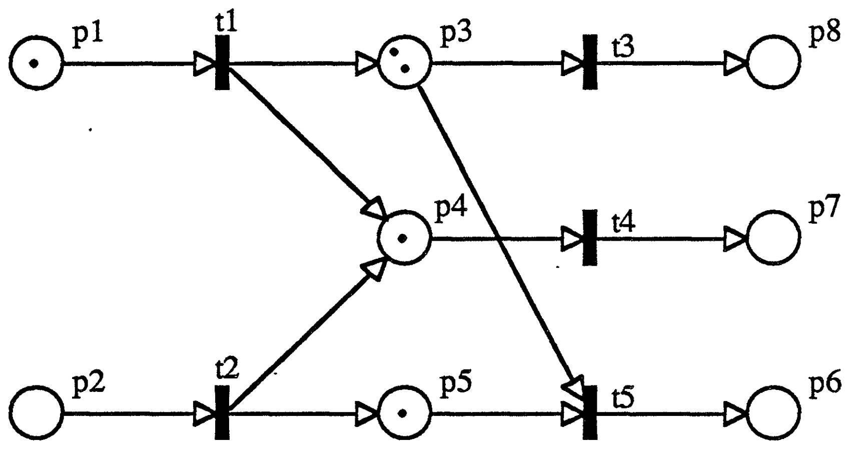


Figure 2.2 Petri Net PN1 with a marking

The execution of a Petri Net is controlled by the number and distribution of tokens in the net. A Petri Net executes by firing transitions. In order to fire, a transition must be enabled. A transition is said to be *enabled* for a marking M, if and only if, for each place of the net:

$$\forall p \in P, \qquad M(p) \geq I(p, t) \qquad (2.1)$$

The transition of an ordinary Petri Net fires by removing from each of its input places one token, and by adding in each of its output places one token. When an enabled transition fires, the resulting marking, M', is obtained from the previous one with the following formula:

$$\forall p \in P, \qquad M'(p) = M(p) + O(t, p) - I(p, t) \qquad (2.2)$$

Figure 2.3. shows the resulting marking of PN1 after transition t5 has fired (initially the marking of the net is the one depicted in Figure 2.2).

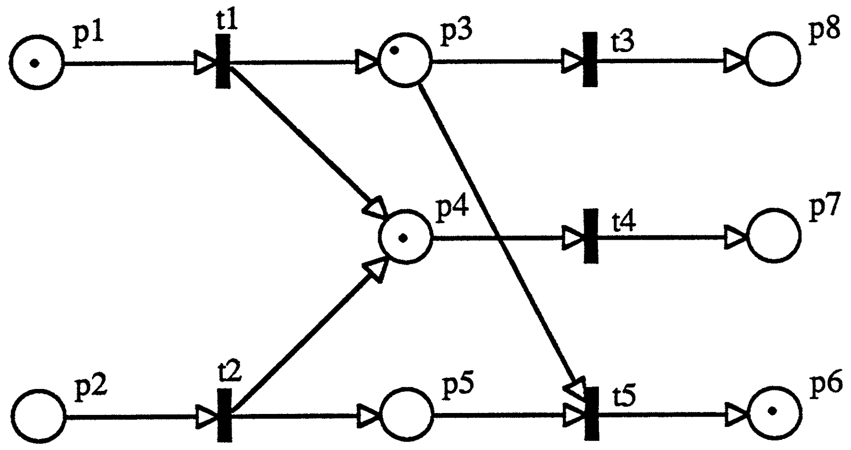


Figure 2.3 Petri Net PN1 after firing t5

2.1.2 Algebraic Representation of Petri Nets

The structure of a Petri Net can be represented by an integer matrix, C, called the *incidence matrix*. The elements, C_{ij} of the incidence matrix are defined as follow:

$$\forall\ (i, j) \in [1, n] \times [1, m],\ C_{ij} = O(t_j, p_i) - I(p_i, t_j) \qquad (2.3)$$

The incidence matrix of the Petri Net PN1 is given below:

$$C(PN1) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & -1 \\ 1 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

In the case of ordinary Petri Nets, C_{ij} takes values in {-1, 0 1} only. It should be noted that self loops cannot be accounted in this representation. As a matter of fact, if t_i and p_j form a self-loop, then we have I (p_j, t_i) = 1 and O (t_i, p_j) = 1, and therefore $C_{ij} = 0$. Consequently, the incidence matrix represents uniquely only ordinary, pure Petri Nets.

A *firing sequence* defines a sequence of ordered firing. It is represented by $\sigma = t_i, t_j, t_k$... This means that t_i fires first, then t_j , then t_k. To each firing sequence σ_s is associated an integer vector N_s. N_s has a dimension equal to the number of transitions, m. The ith element of N_s denotes the number of occurences of transition t_i in the sequence σ_s. If σ_s is a firing sequence, N_s the vector associated with σ_s , and M_0 the initial marking, the new marking M_1, resulting from the firing sequence σ_s, is given by:

$$M_1 = M_0 + C \cdot N_s \qquad (2.4)$$

Definition 2.5:

Given an initial marking M_0 of the net, a *reachable marking* is any possible marking of the net which can be obtained from the initial marking. In other words, given a net and an initial marking M_0 for this net, M is a reachable marking if and only if there exist a firing sequence σ , such that:

$$M_0 \xrightarrow{\sigma} M \qquad (2.5)$$

or

$$\exists\, N_s \text{ such that} \quad M = M_0 + C \cdot N_s \qquad (2.6)$$

The linear algebraic approach is very interesting for analysis. However, this approach should be used with caution in order to avoid its two main pitfalls:

- as noted previously, self-loops cannot be accounted in the matrix representation of the net.
- There is no sequencing information in the firing vector representation of a firing sequence. The firing vector and its use does not provide information as to the order of the firing. It may be possible to obtain a new marking after applying (2.4) with an infeasible firing vector.

2.2 INVARIANTS

2.2.1 Definitions

Definition 2.6:

For a Petri Net described by an incidence matrix C, an S-Invariant is an n-dimensional vector, X, where n is the number of places in the net, such that

$$X^T \cdot C = 0.$$

The ith element of the vector X corresponds to the ith place. The value of each element is a weight attached to the token content of the related place; it is a non-negative integer. The invariance property represented by an S-Invariant, which justifies its name, is that, for any firing sequence, the weighted sum of the token content of the places is constant. Therefore, for an initial marking M_0 and some reachable marking M, we have:

$$\sum_{i=1}^{n} M(p_i) \, . \, x_i = \text{constant} = \sum_{i=1}^{n} M_0(p_i) \, . \, x_i \qquad (2.7)$$

or

$$X^T \cdot M = X^T \cdot M_0 = \text{constant} \qquad (2.8)$$

From relation (2.4), we also have for any firing vector assciated with a firing sequence:

$$X^T \cdot M = X^T \cdot M_0 + X^T \cdot C \cdot N_s \qquad (2.9)$$

By comparing (2.8) and (2.9), we obtain, for any firing vector:

$$X^T \cdot C \cdot N_s = 0 \qquad (2.10)$$

Since (2.10) holds for any firing vector N_s, it follows that:

$$X^T \cdot C = 0 \text{ or } C^T \cdot X = 0 \qquad (2.11)$$

Similarly, a T-Invariant, which can be considered as the dual of an S-Invariant, is a vector with dimension equal to the number of transitions, m. The elements of this vectors are also non-negative integers. **Each integer attached to a transition is such that, for any place of the net, the sum of integers assigned to its input transition is equal to the sum of integers assigned to its output transition.**The condition characterizing a T-Invariant is given by:

$$C \cdot Y = 0 \qquad (2.12)$$

Definition 2.7:

The *support* of an invariant is the set of nodes whose corresponding components in the invariant is strictly positive. The notation for the support of the invariant X is <X>.

Definition 2.8:

The support of an invariant X, <X>, is said to be *minimal* relative to a set of invariants $A = \{X_1, X_2, X_3,.... X_n, X\}$ if and only if it does not contain the support of another invariant but itself and the null vector.

2.2.2 Properties of S-Invariants

Theorem 2.1: (Memmi, 1979)

Let <X> be the support of an S-invariant, and $< X_1 >, < X_2 >, < X_3 >,...,< X_n >$ the minimal supports of S-Invariants contained in <X>, then:

- $<X> = \bigcup_{i=1,..n} <X_i>$
- for any S-Invariant X_j such that $< X_j > = <X>$, there exists n positive rational coefficients λ_i, $i \in [1..n]$ such that:

$$X_j = \sum_{i=1}^{n} \lambda_i X_i$$

where X_i is an S-Invariant whose support is $< X_i >$.

Corollary 2.1:

Let $< X_1 >, < X_2 >, < X_3 >,\ldots,< X_n >$ be all the minimal supports of S-Invariants of a Petri Net (they are necessarily finite as the number of places is finite). Let X_j be an S-Invariant whose support is $< X_j >$. Then, there exist positive rational coefficients λ_i, $i \in [1..n]$ such that:

$$X_j = \sum_{i=1}^{n} \lambda_i X_i$$

This corollary is the basis for an algorithm which finds all the minimal support S-Invariants, (Alaiwan et al., 1985; Martinez et al, 1980).

Example:

$X = (x_1, x_2, x_3, x_4, x_5, x_6)^T$ is an S-Invariant of the net PN1 depicted in Figure 2.1. if and only if it satisfies equation (2.11), which yields:

$x_1 = x_3 + x_4$

$x_2 = x_4 + x_5$

$x_3 = x_8$

$x_4 = x_7$

$x_3 + x_5 = x_6$

From these equations, the reader should be easily convinced that there are three distinct minimal support S-Invariants:

$X_1 = (1, 1, 0, 1, 0, 0, 1, 0)$ with $<X_1> = \{p1, p2, p4, p7\}$

$X_2 = (0, 1, 0, 0, 1, 1, 0, 0)$ with $<X_2> = \{p2, p5, p6\}$

$X_3 = (1, 0, 1, 0, 0, 1, 0, 1)$ with $<X_3> = \{p1, p3, p6, p8\}$

Definition 2.9:

Let X be a S-invariant and <X> its support. <X> is a set of places which is a subset of P. A S-component associated with X, and denoted [X] is a subnet of the Petri Net whose set of places is <X> and whose transitions are the input and the output transitions of the places of <X> in the Petri Net. We have therefore the following:

$[X] = (P_x, T_x, I_x, O_x)$ with:

- P_x = <X>.
- T_x = {p· such that p belongs to <X>} and { ·p such that p belongs to <X> }, where p· denotes the output transitions of p and ·p denotes the input transition of p.
- I_x is the restriction of I to P_x x T_x.
- O_x is the restriction of O to T_x x P_x.

The same type of definition applies to a T-component.

2.3 PROPERTIES OF PETRI NETS

2.3.1 Liveness

Definition 2.10:

A marking M_0 is live if and only if, for any transition t, and for any reachable marking M from M_0, there exists a firing sequence from M_0 which fires t. In other words, this property guarantees the firing process to be deadlock free.

2.3.2 Boundeness

Definition 2.11:

A marking is bounded if there exist a positive integer, N, such that for any marking M, the number of tokens in each place is bounded by N.

Theorem 2.2: (Memmi, 1979)

If there exist an S-Invariant which covers the net (i.e. which support contains all the places of the net), then the net is bounded for any finite initial marking.

Corollary 2.2:

If any place of a Petri Net belongs to at least one minimal support S-Invariant, then the net is bounded.

By applying theorem 2.1, we know that a linear combination of all the minimal support S-Invariants of the net will provide another S-Invariant. As any place of the net belongs to at least one minimal support S-Invariant, then the newly created S-Invariant covers the net, and therefore the net is bounded.

This powerfull result is unfortunately partial as the converse is false: if a net is not covered by an S-Invariant, this does not necessarily means that it is not bounded. Figure 2.4 depicts a net which is bounded, but not covered by S-Invariants. As a matter of fact, this net has one S-Invariant with a distinct support:

$X = (1, 1, 1, 0)$ with $<X> = \{p1, p2, p3\}$.

The net depicted in Figure 2.4 also shows that S-Invariants may not be that trivial to find, even with the powerful graphical representation of Petri Nets.

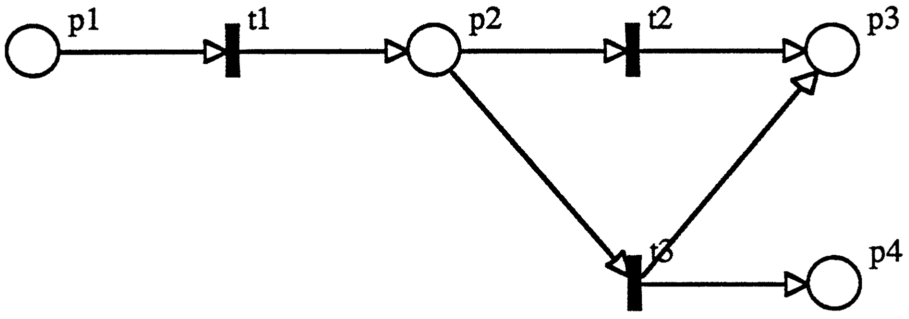


Figure 2.4 A bounded Petri Net not covered by S-Invariants

The general result, Memmi (1979), is that a Petri Net is bounded if and only if there exist a vector I of strictly positive integers such that:

$$C^T . I \le 0 \qquad (2.13)$$

2.4 SWITCHES

2.4.1 Definition

Unlike all the notions introduced in the previous sections of this chapter, switches are not part of the definition of ordinary Petri Nets as found in the general literature. However, Tabak and Levis (1985) have used the concept to extend ordinary Petri Nets, when the full extension to Colored Petri Nets (Jensen, 1987) or Predicate Transition Nets (Genrich and Lautenbach, 1981) and (Genrich, 1987) is not warranted.

Switches are introduced to automate the resolution of conflicts. In the situation presented in Figure 2.5., the token which appear in place p1 can go either to place p2 after the firing of transition t1, or to p3 after the firing of transition t2. The choice is completely arbitrary and there is no mechanism in the Petri Net theory to automate this choice. This is a serious shortcoming. Switches have been specifically introduced to address this problem in a simple manner.

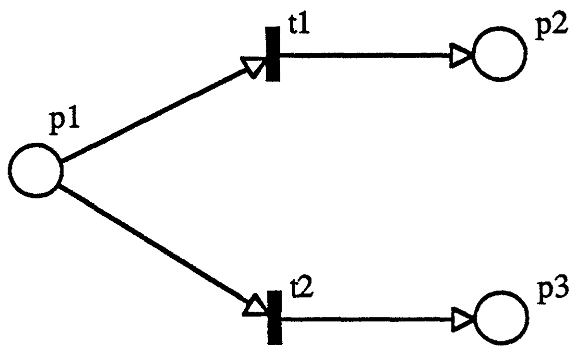


Figure 2.5 Petri Net PN2 with conflict

Definition 2.12:

A switch is a transition with multiple output places and some decision rule, which directs the output flow of information toward one and only one of its output places.

Since a switch is a type of transition, the firing rules for a switch are identical to the firing rules for a transition: a switch needs to be enabled. However, unlike transitions, all the output places of a switch will not receive a token. Only one of them will. This place will be chosen by the internal decision rule associated with the switch. This rule can be deterministic or not. The rule may take the input information implicitly into account or not. Decision rules may be represented by algorithms, but they can also involve echniques derived from artificial intelligence. There is no limitation on the kind of decision rule associated with a switch. Examples of powerful uses of decisions rules associated with switches can be found in Jin, (1985), and Weingaertner (1986).

The resolution of the conflict in Figure 2.5. by a switch is shown by the net in Figure 2.6. A switch is represented by an unfilled rounded bar. Note that the number of nodes and the number of arcs has increased.

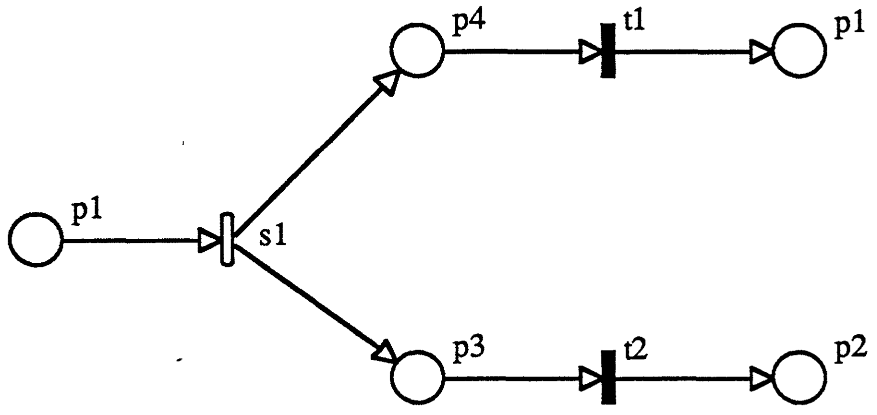


Figure 2.6 Petri Net PN2 with a switch

2.5 APPLICATIONS

Petri Nets are useful for modeling the flow of information and control in decision making systems. In particular, Petri Nets permit to show concurrency or parallelism in systems and to represent the precedence relations between the processing of different components.

The operations executed by the various components may be asynchronous. There need not be global mechanisms that coordinate the scheduling of the processes. Each component usually starts its processing as soon as it has received all the information it needs. If several tasks are requested from the same component, a queuing discipline exists to prioritize the various requests. Such discipline may be of the FIFO (First In First Out) or LIFO (Last In First Out) type (Grevet, 1988).

To model the dynamic behavior of a system, the execution of a process (or task) is represented by the firing of a corresponding transition. The flow of tokens represents the flow of information in the process and the marking of the net at any time represents the state of the system at that time. Liveness and boundedness characterize a well defined system as these two properties guarantee the system to be deadlock free and that information cannot duplicate infinitely at some steps of the process.

The graphical nature of Petri Nets helps to visualize the complexity of the system. Therefore they are of use to the analyst as well as to the decisionmaker. The Petri Net formalism is a good medium of interaction between these two groups of people with different perspectives and interests in the design and evaluation process of distributed systems.

A distributed system represented by a Petri Net receives information from the environment (sources), which is represented by places, and produces responses which appear at the output nodes (sinks) which are also represented by places. Such a system, if it does not contains loops, is represented by an *acyclic* Petri Net in which where there are places with no input transitions (sources), and places with no output transition (sinks). A simple path is then defined as a directed path which starts at a specified source and ends at a specified sink.

A marked graph is a Petri Net such that any place has one input and one output transition only. It is possible to consider a degraded form of a marked graph where the sources would not have an input transition, and where the sinks would not have an output transition. Provided cautious actions are taken, most of the results available for marked graph can be extended to this particular form of marked graph. An important result is that, in the case of a marked graph, simple paths and S-components coincide. This is because when a token is in a place belonging to a path, it cannot get away from this path, as a place has one input transition and one output transition only.

CHAPTER III

MODELING FUNCTIONALITY IN DISTRIBUTED SYSTEMS

3.1 INTRODUCTION

A good model of a distributed system should describe precisely specific properties or characteristics of the system under consideration. The model should specify as much as possible, as what is not specified cannot be checked. This means that as much as possible of the features relevant to the focus of the study should be incorporated into the model. Models of the same system may differ according to the specific problem they address. Depending on the focus of the analysis, there may be more emphasis put on certain aspects of the system than others. Therefore, depending on the study to be conducted, the level of detail of the description of certain aspects of the system changes.

Several methodologies have been developed to integrate all the relevant specifications or properties of a distributed system in one approach. Among these, the MCES methodology, described by Sweet (1986), is one of the most promising. This methodology resolves the conflicts between the different constituencies involved in the process, and is made of seven distinct steps. The MCES methodology is also a structured top-down way to evaluate a system's performance.

However, the methodology remains descriptive and is qualitative to a large extent. A quantitative method needs to be integrated. In the past several years, the System Effectiveness Analysis (SEA) approach has been developed by Dersin and Levis (1981), and then extended by Bouthonier and Levis (1984), Cothier and Levis (1986), Karam and Levis (1985), and Martin and Levis (1986). This integration is possible because both approaches are based on the same basic concepts.

There are six common concepts. These concepts are: **system, environment, context, parameters, measures of performance,** and **measures of effectiveness.** The first three elements permit to pose the problem adequately, while the last three define the quantities of the problem.

A system consists of components, their interconnections and a set of operating procedures. From the list of components and their interconnections, a boundary can be drawn that defines what is, and what is not, included within the system.

The environment consists of assets that are related to but are not part of it, and other assets on which the system's related assets can act upon. The system can sense the environment, and eventually acts upon it. On the other hand, the environment can also act upon the system. A typical example is one in which a distributed system is used to monitor the environment and to direct the actions of assets (Early Warning System).

The context denotes the set of assumptions and conditions within which the system and the environment exist. The context has an influence on the system, but the system cannot act upon the context. Figure 2.3 depicts the relationship between these three entities.

Parameters are quantities that specify the system and its goal. In the case of Air Traffic Control, the system's parameter are such quantities as detection radius of the radars, the computational time delays of the computers, the screen update rates, the size of the different control sectors, the failure probabilities attached to each component, and many more. The system's parameter are defined within the system's boundaries. The parameters of the mission are the interarrival rate of the airplanes to be monitored, their speed, and the size of the area to be controlled.

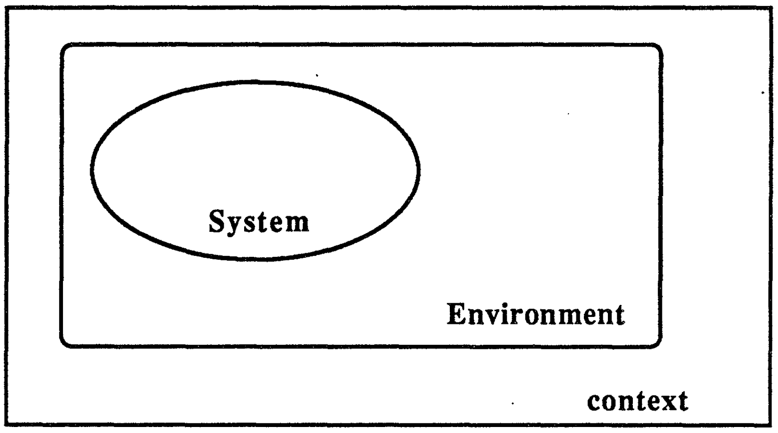


Figure 3.1 Relationship between System, Environment, and Context

Measures of Performance (MOPs) are quantities that describe system properties or mission requirements. MOPs in our example may include reliability, cost, or probability of accurate detection. Both the system's requirements and the designed or implemented system should be described by the same MOPs. Lastly, MOPs may include for their evaluation some of the aspects of the environment.

Measures of Effectiveness (MOEs) are quantities that result from the comparison of the system's requirements and the proposed or implemented system's MOPs. They reflect the extent to which the proposed or implemented system meets its requirements. It is necessary to consider the effect the system has on the environment to evaluate the MOEs.

3.2 MODELING THE REQUIREMENTS

In this section, a method is described for representing the functional requirements of a system. The method uses the Petri Net formalism, and incorporates part of the MCES methodology, more precisely, its first four steps.

3.2.1 The Concept of Scenario

At the beginning of its formulation, the requirements for a distributed system are expressed as a goal, task, or mission to be performed. Given the environment, a set of inputs and outputs are selected and the set of basic functions is defined. A function is a process that treats a set of input and produces a set of outputs.

A scenario is a set of conditions on the environment and context. A given scenario, affects the inter-relationships between functions. Generally, the set of functions needed by the system to perform its task depends on the scenario. In some scenarios, a particular function may not be needed while in another scenario it may become crucial. Therefore, as the scenario changes, the set of basic elements needed by the system to handle the task changes. As a result, the links between the different components of the system vary accordingly. This introduces the notion of variability. For this type of variability, the system adapts its structure to the changes in the environment. The system may adapt in two different ways:

- The state of the environment may impose the use of a different set of functions so that the task can be handled in the most appropriate way. For example, in an Air Traffic Control System, the function "provide emergency services" may have to be used for certain states of the environment (a plane in emergency landing, for example) while this function is not needed most of the time.
- the system may use equivalent functions, that is functions whose conribution to the overall task is equivalent, but whose processing means are different. This may require the use of different devices or different protocols. For example, in a defense system, an

identification mission can be handled either by a plane or by a radar, according to the state of the environment.

This dependence of the processing structure and elements of the system on the scenario is type 2 variability, as defined by Monguillet (1987). As a matter of fact, type 2 variability is defined as change in the structure of a system according to changes in the environment, as perceived by the system.

Since we can define the functional requirements of a system only in the context of a mission or goal and for a given scenario, and since we can represent the functioning of an implemented system only for a given scenario, the representation of both the requirements and the actual implemented system in this work will be done for a particular scenario.

3.2.2 The Concept of Strategy

Once a scenario is given for which the requirements are to be modeled, the components and the interrelationship between these are fixed. However, this does not mean that the course of action is fixed. There is still another type of variability which can be encountered: this variability is due to the course of action and is based on the notion of strategies employed by the system to accomplish its mission. Each strategy activates a set of functions and a set of specific interrelationships. A typical case occurs when the input data to be processed by a function are deemed insufficient so that a query for additional information may be made first. In the event that the data are deemed sufficient, then there is no query for additional information.

The choice of strategy deals with the inputs of the system. To embed the different strategies of a system in its representation, switches are used. Switches have been described in chapter II. A switch will be used when a decision about which action to pursue that leads to the accomplishment of the mission (or goal or task) is to be made. A switch may also be used to represent a meta-decision, that is a decision about a decision or deciding how to decide. Therefore, a variable structure is implicitly created in the representation of the system. This variability is of type 1, as defined by Monguillet (1987). Type 1 variability is characterized by the adaptation of the processing structure to the input being processed. Figure 3.2. shows an example of a representation with a switch.

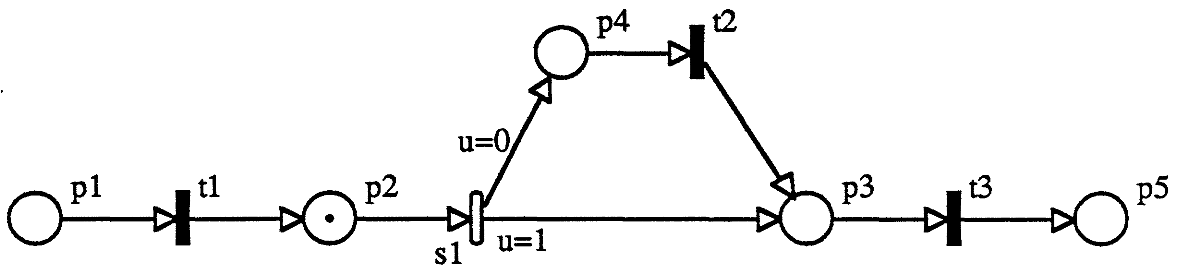


Figure 3.2 Petri Net PN3 with a Switch

In this net, two alternative strategies can be implemented. According to the information embedded in the token in p2, either more information will be requested (p4, t2) or the data are sufficient so that the processing can be continued by t3 directly. In this representation, the switch s1 represents the implementation of the two different strategies. To choose the most appropriate strategy, a decision rule is embedded in the switch. The decision rule is given by the value of the strategy u. In this case, u can take the value 0 or 1. If a token is present in place p2, then s1 is enabled and can fire. If u is set to 0, then a token is added to p4, while if u is set to 1 a token is added directly to p3.

Other cases may arise where the courses of actions are changed by the implementation of a strategy. It may well happen that when a strategy is chosen, the resulting processing may be completely different than if another strategy had been chosen. Such a case is depicted in Figure 3.3. In this representation, a first switch, s1, permits to choose between two distinct courses of

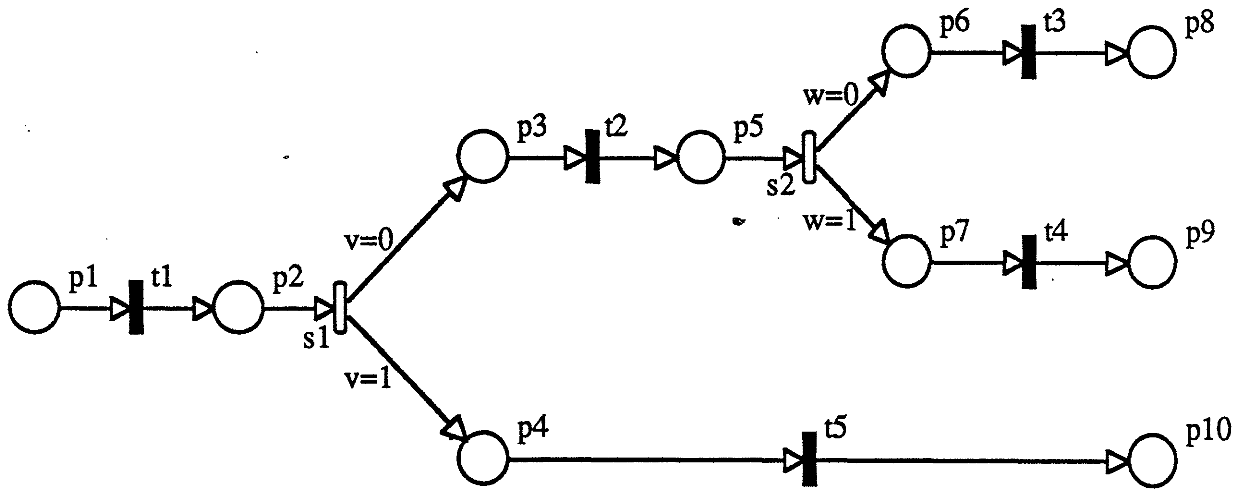


Figure 3.3 Petri Net PN4 with two Switches

actions. In the event that v equals 0, then another set of alternative courses of action is offered. According to the value of w, either 0 or 1, one out of the two possible alternative courses of actions will be chosen.

In the Petri Net representation of the requirements, there may be more than one switch, especially if the system is to be used in a scenario where the states of the environment are numerous. Consider a Petri Net with N switches, each switch with N_i output places. The maximum number of ordinary Petri Nets obtained is therefore:

$$\prod_{i=1}^{N} N_i \qquad (3.3)$$

This maximum number of Petri Net may not be reached. Two different cases may be distinguished which explain this phenomenon:

- the case of cascaded switches, that is when one output place of a switch, p1, is connected by a directed path to the input of another switch. In such a case, if the output of the first switch is not p1 there is no need to worry about the setting of the second switch as it will be inactive. This is the case of the net depicted in Figure 3.3.
- the case of mutual constraints for the setting of two switches. For example, the setting of switch A may be related to the setting of switch B. In this case, the number of combinations of settings of the two switches is reduced.

In the Petri Net representation of the requirement, each source of the net will represent the necessary input for a function rather than the output of a particular type of sensor (radar, sonar, weather forecast report), or a particular means of transmission (radio HF, radio VHF, telephone, computer network, etc...).

The Petri Net representation obtained incorporates switches which represent all the possible courses of actions. The ordinary Petri Net formalism does not include switches in its grammar. Therefore, this representation must be transformed into an ordinary Petri Net representation, so that all the analytical tools of the formalism can be applied.

To do that, switches should be removed. To remove the switches, each switch should be first assigned a value of its decision rule. When this is done, the connectors from each switch to the other output places of the switch which have not be chosen can be suppressed. Therefore, each switch is transformed into a regular transition. An ordinary Petri Net is then obtained.

On the Petri Net PN3 of Figure 3.2, the procedure is applied in the case where u takes value 1. The resulting net obtained by removing the connector from S1 to p4 is depicted in Figure 3.4.

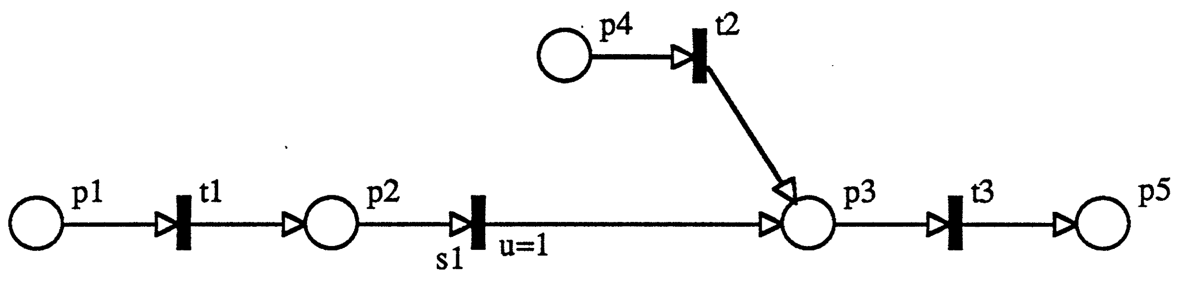


Figure 3.4 Petri Net PN2 in the case where u =1

However, the figure shows that the removal of the connector has made p4 take the form of a source. In general, if there are no other inputs to the place p4, as is the case here, p4 should be eliminated too. The same argument is applied to t2, which is also eliminated. The basic idea is to remove all nodes of the paths starting from the unused output nodes of the switch until a node which is included in another path of the net is encountered. This means that all nodes of the path should be removed until a node with multiple inputs is encountered, with at least one of these inputs being a node that does not belong to such paths. The algorithm to remove the nodes in the case the net has sources is presented below.

```
BEGINNING
    choose a switch setting (for each switch)
    FOR each switch DO
        remove connectors from switch to each not selected output place. There is then only one
        connector left from the switch to one of its output places.
    END
    FOR each source DO
        find all simple paths from the source
        FOR each path DO
            mark all nodes included in the path
        END
    END
    remove all not marked nodes
END of algorithm.
```

Once a switch setting has been chosen, the nodes to be kept are marked, while the others are removed. This algorithm takes advantage of the directed nature of the Petri Net. The application of this algorithm is demonstrated on a more complex example depicted in Figure 3.5. This net, which does not necessarily model the requirements for a distributed system, contains three switches.

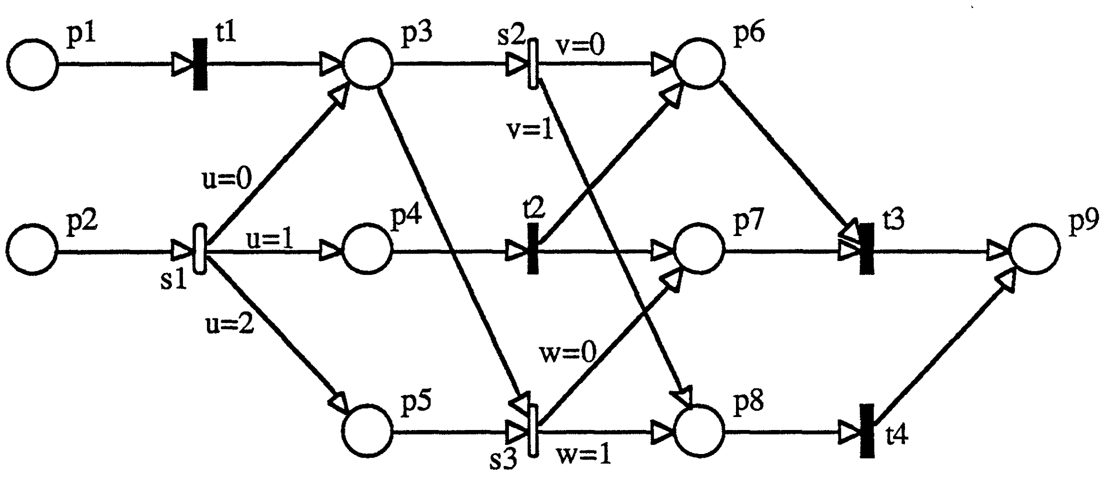


Figure 3.5 Petri Net PN5 with Switches

If it is supposed that one strategy requires s1 to be set to u=2, s2 to be set to v=1, and s3 to be set to w=0, the net depicted in Figure 3.6 is obtained. In this figure, the simple paths originating at the sources, and the marked nodes described in the algorithm are shown in heavy lines .

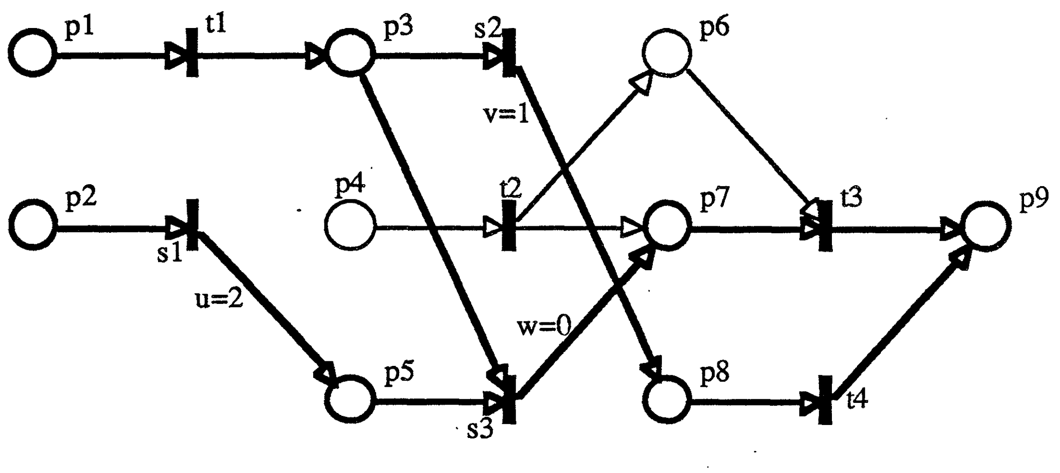


Figure 3.6 Petri Net PN5 with u=2, v=1, and w=0

The last step is to remove the non marked nodes, (p4, t2, p6) which leads to the net in Fig. 3.7.

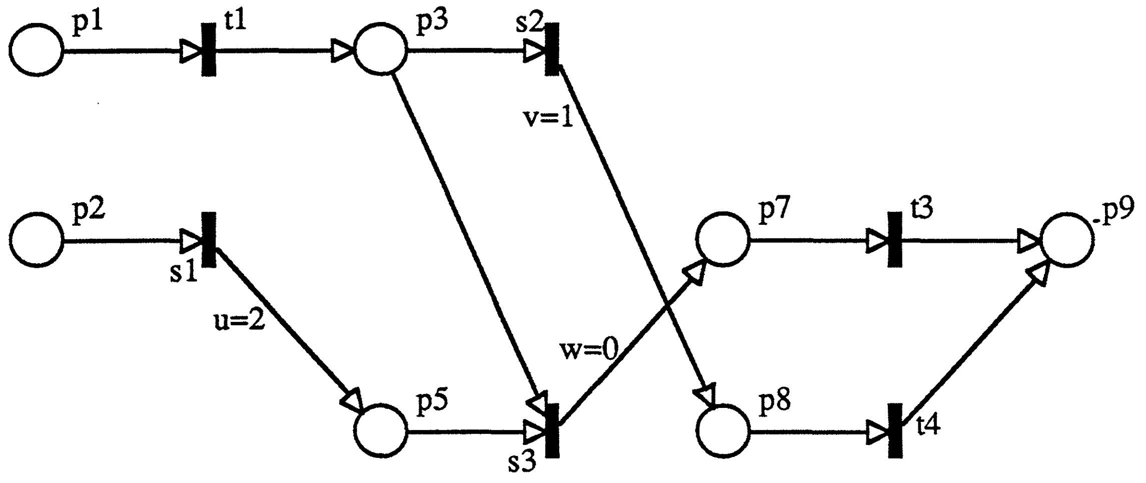


Figure 3.7 Petri Net PN5 with unnecessary nodes removed

The implementation of this algorithm can be made as a breadth-first or depth-first search originating at each source of the net. Other applications of breadth-first or depth-first search can be found in Grevet (1988). Since the connectors between each switch and its unused output places are removed, the effect is the same.

For each strategy, the switches are removed, and we thus obtain an ordinary Petri Net. Therefore, an ordinary Petri Net is obtained for each different strategy. Since each strategy is distinct, the resulting Petri Nets constitute a set.

3.2.3 Requirements Analysis

Once the requirements for the system are expressed in the form of a Petri Net, it is possible to analyze them. The modeler should verify that the Petri Net representation of the requirements matches his/her view of the system. In particular, two problems can be addressed, at this stage.

The pattern depicted in Figure 3.8 should not be allowed in a Petri Net representing the requirements for a distributed system. As a matter of fact, this pattern represents a free choice, i.e., when a token is in place p1, there is no rule attached to the place that permits us to know whether the token will go to transition t1 or transition t2. It is better to replace this pattern by a switch in which a rule is embodied. It may well be that in a particular case the choice between branches is random, but a switch may contain a rule that selects branches randomly.

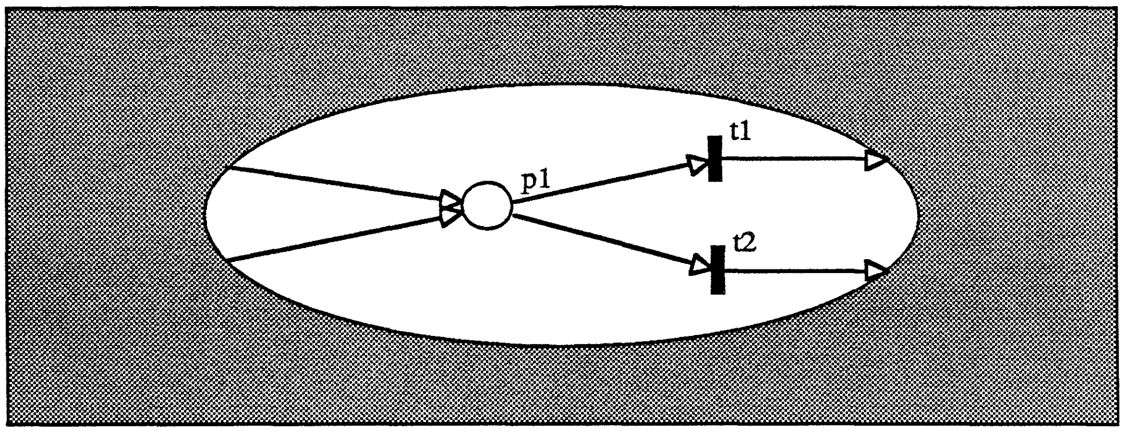


Figure 3.8 Free Choice pattern

The representation of free choice using a switch is depicted in Figure 3.9.

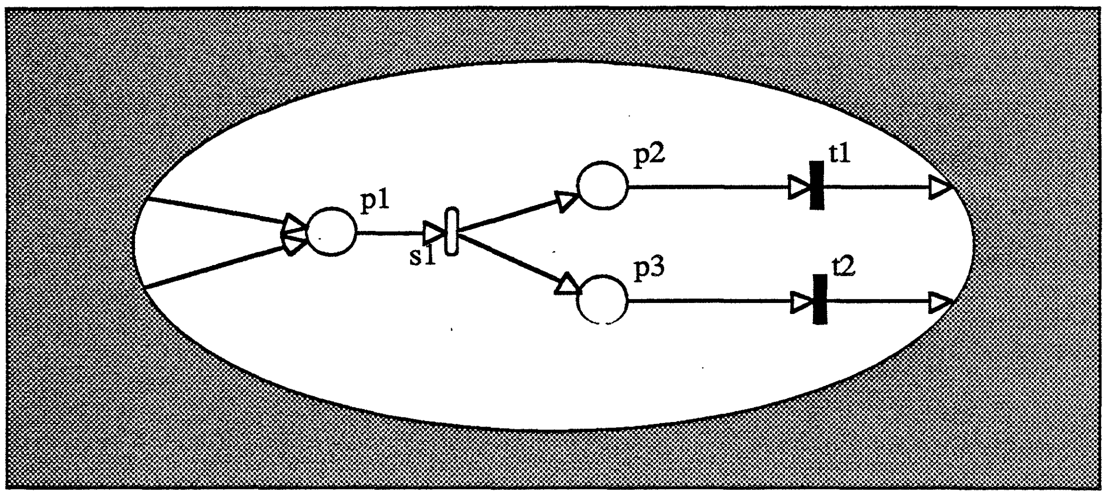


Figure 3.9 Equivalent representation of the Free Choice pattern

A model for the requirements should at least be live and deadlock free. Liveness will ensure that all information will be processed, while a deadlock free net will make sure that all information will go through the required system and will be used to its full extent. When switches are present, particular care should be taken to avoid deadlock. The simple example of Figure 3.10 shows a net with a switch.

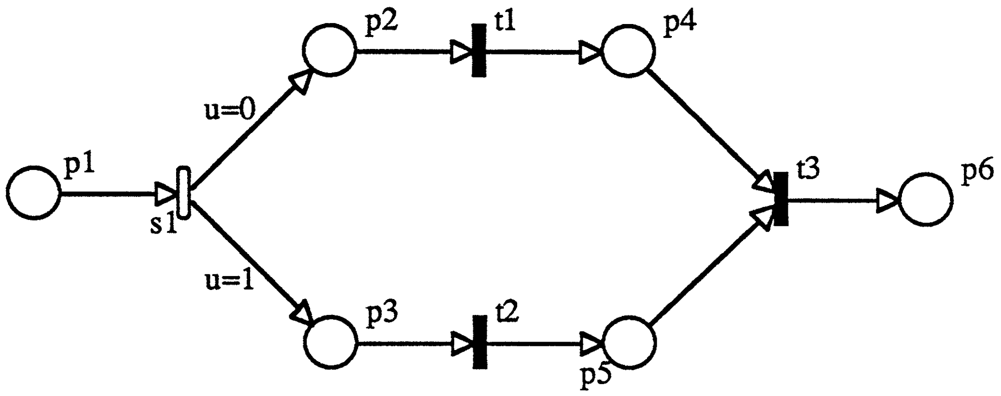


Figure 3.10 Petri Net PN6

If one strategy sets s1 to u=0, then a token is placed in p2 and t1 is enabled. When t1 fires, a token is placed in p4. However, t3 is not enabled because there is no token in p5 and deadlock results. This situation is shown in Figure 3.11, where the sub-net in bold face represents the successive nodes the token passes through before stopping in p4. The only way to resolve this deadlock is for another token to be generated at p1 and then switch s1 be set to u=1 so that a token is eventually generated in p5.

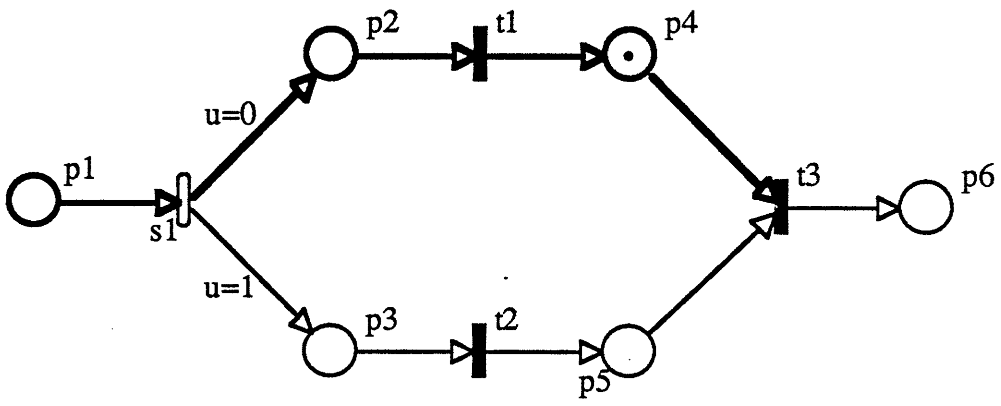


Figure 3.11 Deadlocked Petri Net PN6 when u=0

It should be stressed that the set of Petri Nets representing the functional requirements does not embody any specifications about system performance nor does it specify the degree to which functions are distributed. Lastly, this requirement does not specify the degree of redundancy in the system, if any.

Therefore, these requirements can be viewed as the **minimal functional requirements**, in the sense that they specify the desirable interrelationship between functions and provide a model of the system.

3.3 MODELING THE IMPLEMENTED OR PROPOSED DISTRIBUTED SYSTEM

3.3.1 A Physical System

Besides accounting for the interrelationship between its processes, the model of the implemented or proposed system should account for the geographical dispersion of its physical entities. Thus, the modeling will have to represent the multiple decision and processing agents in their location.

For now, each physical location will be called a shelter or facility. A shelter contains one or more agents together with their supporting systems. Each agent consists of a human and a terminal with its associated software. It may also be an automated processing unit such as a computer used as a server or used as a database.

For now, an agent will be generically identified with a workstation. Each workstation can carry out one or more processes, or functions. Communication systems are used to transmit processed information among workstations. For reliability, or survivability in the case of some systems, there may be a duplication of functions. Therefore, there may exist a number of replications of the same function embedded in the proposed or implemented system. Different workstations may be able to perform the same process.

3.3.2 Modeling Functionality

Since distributed systems receive information from different sensors and sources, it follows that the model of the real system should incorporate a large number of inputs. In this work, each source in the net represents a type of message without discriminating on the nature of the sensor or the means used to transmit the information. In other words, a source represents the place where a specific type of message arrives. This arbitrary definition of source is sufficient for a functional analysis of the system. However, in this frame it is not possible to represent accurately different communication and data networks, each corresponding, for example, to a different level of security or access privilege.

Among these input signals received by the system, some of them may trigger some part of the processing that the system can do, while others may trigger other parts of the system. To handle this, temporary storage is needed and a database model is needed.

In previous work, Bejjani (1985) has modeled databases with Petri Nets. It is outside the scope of this work to study in great detail the database model. However, there is need to understand the interrelationships between the processes. Output signals of some processes are stored in databases. Each different output signal is stored in a different database.

Figure 3.12 shows a Petri Net model of a database. The information to be stored arrives in place p1. This is modeled by a token created in p1. The token enables the output transition of p1, which represents the operation of storing the information in the database. When this is done, p2 contains a new token which represents the new stored information. These data are now available upon request from a process in the system. The request for data stored in this database is materialized by a token in place p4. If any data is stored in the database, i.e., if there is at least a token in place p2, then the transition t2 corresponding to the acquisition of the data can fire, and a token is created in p5. When t2 fires, a token is created in p3, which enables t1, which by firing creates a token in p2. This loop is created to model the availability of the data for another query. However, as times passes, new data may be stored in the database, that is, updates may occur.

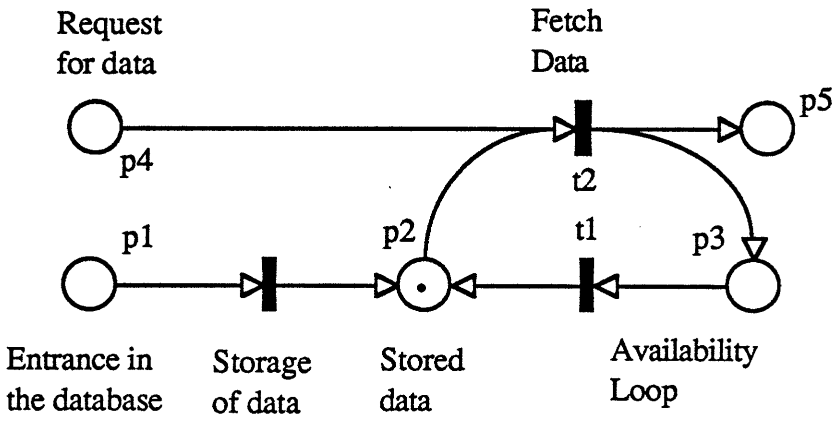


Figure 3.12 Petri Net model of a Database

This raises the need for protocols, since in this case there is more than one token in p2 and thus a problem exists as to which token to be considered for the firing of t2. A LIFO protocol (Last In First Out) solves the problem since in this case the last update is the best one to use.

In order to perform his functions adequately, an agent needs some resources. An agent may need the use of decision aids, expert systems (Perdu 1987), graphical displays, etc. For example, in the case of an operator performing the basic function "airplane identification" in an Air Traffic Control system, this agent needs some resources, namely a radar screen, and the access to a computer and its software.

Resources are modeled with loops in a Petri Net. Each physical resource is represented in a Petri Net by a place that describes the availability of that resource. A resource loop from one resource place to a particular agent performing a certain role indicates the need of that resource. For reasons discussed in section 2.1.2., these loops contain more than just one place and one transition. The marking in the resource loop indicates whether the resource is available or not. Figure 3.13 provides an example of the modeling of a resource. The process represented in the Petri Net by transition t2 needs a physical resource. This resource has its availability, from t2's point of view, indicated in the output place of t4. The presence of a token in this place means that this resource is now available for t2.

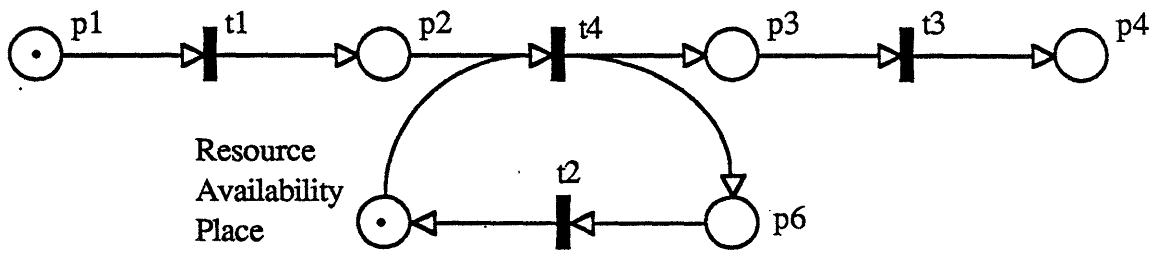


Figure 3.13 An example of a Resource Loop

However, in the model of the system presented in this work, although functions need resources, there will not be any representation of the resources. It will be assumed that the access to the needed resources has been checked, and the availability of these resources is implicitly embodied in the transition representing the process using that particular resource. Therefore, the problem of the availability of shared resources is not addressed in this work.

3.3.3 Modeling the Implemented System

The basis for representing the proposed or implemented system is the construction of a series of diagrams that describe the system, as designed, and its operations. The first diagram shows the localization of functions. The functions are identified and their allocation to the various workstations that constitute the system is determined. Given that the system is distributed, and that it consists of a number of different shelters or facilities, then the specification of which facility houses each workstation is included. The diagram can be described as follows:

Functions ---> Workstations ---> Shelters

The second diagram traces the input flows. The various sources of input messages are identified and the various messages they generate are specified. Then the individual messages are tracked to all the system components and workstations by which they are first processed. The diagram shows the flow of individual message from sources to the first component.

The third diagram traces the output messages, whether commands or information, and the internal displays that are available to selected nodes of the system. In this diagram, all the output messages and their destinations are identified and the workstations or components that produce them are specified.

The fourth diagram deals with the communication network needed:

- between the system and its environment
- between the shelters
- within the shelters, between the workstations

The fifth diagram depicts the interrelationship of functions. Here, there is not yet the notion of scenario involved. Therefore, the relationships between functions are multiple as the specific relationship between functions might change for different scenarios. Further, it should be noted that these interrelationships impose constraints on the information flows through the network.

When these diagram are acquired, it is possible to model the information flows. Different partial information flows can be obtained according to the point of view taken. From the message point of view, a flow graph can be derived for each message that shows all its critical paths. From the workstation point of view, a flow graph can be derived that shows all the messages necessary to perform each function.

All these graphs can be aggregated to form a Petri Net with switches. The aggregation is done by superposition of all the graphs of the same type. The result is a Petri Net with many connectors and switches. Switches model different aspects of the system:

- As with the requirements, switches may be introduced to model the different possible strategies within a scenario.This accounts for type 1 variability.
- Switches can also model the choice of a functional path to be followed according to the state of the context, that is, the scenario at hand. This time, these switches account for type 3 variability as defined by Monguillet (1987). Type 3 variability defines the cases

where the system adapts its structure to changes in the system's parameter. For example, in case of failure of some components, it may still be possible to accomplish the mission by reconfiguring the system and using other paths from the sources to the sinks.

3.3.4 Representation of the Functionality in the Frame of a Scenario

In the next step, the unnecessary functional paths for a chosen scenario are removed. To do that, we need to consider all input messages that are not relevant for the chosen scenarios and to eliminate them. This will alleviate part of the complexity of the representation. The elimination of the irrelevant nodes can be cumbersome because some of the nodes that are part of an unused simple path may still be crucial for the completeness of the representation of the net.

In the end, a Petri Net with switches is obtained along with a table of switch settings which allow to convert this net into a set of ordinary Petri Nets. These Petri Nets are **mutually exclusive.**

3.4 REPRESENTATION OF FUNCTIONALITY

3.4.1 Introduction

In this section, the structure of a Petri Net model of a distributed system is related to the functionality of such a system. This relation relies on the analysis of the net by minimal support S-Invariants. After reviewing some graph theoretic definitions of interest the relation between the structure of the net and the functionality of the modeled system is presented.

The Third Webster International Dictionary states that functionality is the quality, state or relation of being functional. Among the numerous definitions, "functional" is defined as:

1. to contribute to the development or maintenance of a larger whole:
   having a useful function.
   This definition is further broken down into several different aspects:
   - designed or developed from the point of view of use
   - carrying out or consisting of a group of related activities: performing a specialized service.
2. performing or able to perform its regular function: in a functioning condition.
3. relating or attempting to demonstrate the relatedness of any single aspect to the maintenance of a whole.

Each definition addresses a different aspect of functionality. A function can be defined as a group of related actions contributing to a larger action, i.e., it can be viewed as a process. In this work, we will assume that each function is a well defined algorithm. Therefore, a functionality can be described as a set of processes, or functions, executing a more global action when used together in an ordered way. This definition does not state very precisely what is meant by "ordered way". This is deliberate. There may be several sequences of the different processes that lead to the execution of the same global action. Furthermore, the set of necessary processes may not be unique. There may be different choices or combinations of functions leading to the same functionality.

In the rest of this section, the structural properties of a system, as described by a Petri Net, are related to the functions that a system performs. Some sub-nets of interest are identified. Then algorithms are presented to determine these sub-nets. To do that, the theory of S-Invariants is used. Therefore, after some necessary graph theoretic definitions, properties of S-Invariants are presented.

3.4.2 Graph Theoretic Definitions

Definition 3.1:

A *directed path* in a Petri Net is a sequence of k nodes, where k is at least 2, and k-1 connectors such that the ith connector connects the ith node to the i+1st node.

Definition 3.2:

A directed path is *simple* if and only if all its connectors are distinct.

Definition 3.3:

A directed path is *elementary* if and only if all its nodes are distinct.

It should be noted that any elementary path is a simple path, while the converse is false. The Petri Nets shown in Figure 3.14 provide examples of the different types of directed paths.

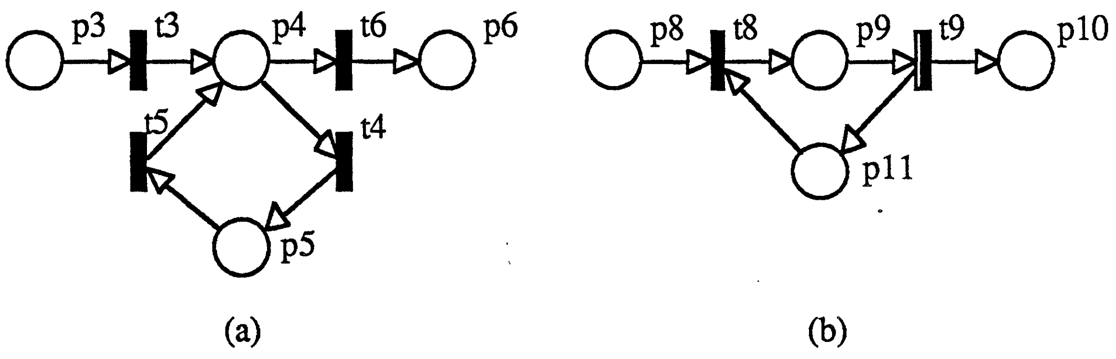


Figure 3.14 Examples of Directed Paths

In the Petri Net of Figure 3.14 (a), {p3, t3, p4, t6, p6} is a directed path. {p3, t3, p4, t4, p5, t5, p4, t6, p6} is also a directed path, a simple path but not an elementary path. In the Petri Net of Figure 3.14 (b), {p8, t8, p9, t9, p11, t8, p9, t9, p10} is a directed path, but not a simple path because the connector between t8 and p9 is repeated twice in the sequence. Therefore, this path is not an elementary path.

Definition 3.4:

A *circuit* is a directed path such that its first and last nodes coincide.

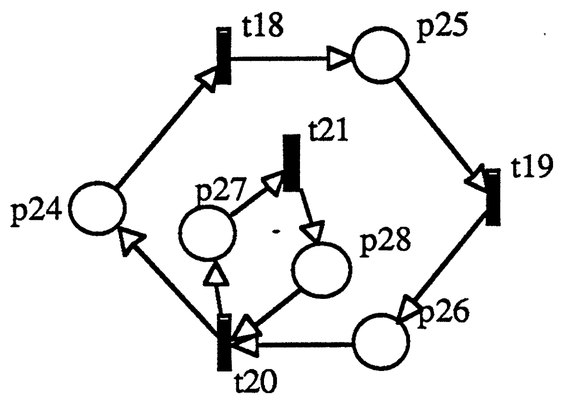
Definition 3.5:

A circuit is *simple* if and only if its path is simple.

Definition 3.6:

A circuit is *elementary* if and only if its path is elementary, except for the last node which is, by definition, identical to the first one.

The Petri Nets shown in Figure 3.15 provide examples of the different types of directed paths.



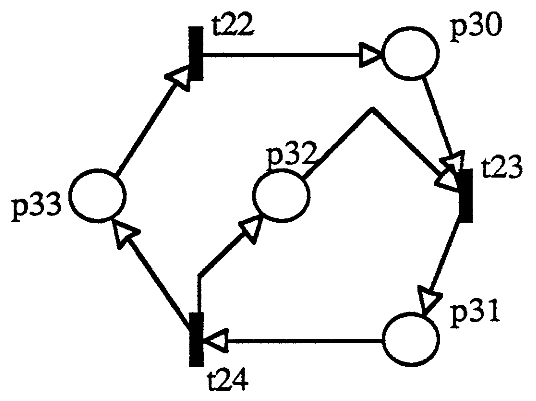


Figure 3.15 Examples of Directed Circuits

In the Petri Net of Figure 3.15 (a), {p24, t18, p25, t19, p26, t20, p27, t21, p28, t20, p24} is a simple circuit, and not an elementary circuit. On the other hand, {p24, t18, p25, t19, p26, t20, p24} is an elementary circuit and, therefore, also a simple circuit. In the Petri Net of Figure 3.15 (b), {p30, t23, p31, t24, p32, t23, p31, t24, p33, t22, p30} is a directed circuit, it is not a simple circuit and, therefore, not an elementary circuit.

3.4.3 Information Flow Paths and Functionality

In this section the structural properties of a system, as described by a Petri Net, are related to the functions that a system performs. Some sub-nets of interest are identified.

Definition 3.9:

In a Petri Net model of a distributed system, a *simple information flow path* is any directed path from a source to a sink.

A simple information flow path represents a token movement from a source to a sink. Starting from the source, an ordered chain of transitions allows, by a succession of firings, to remove that token from the source and create a token in the sink. In the process, a sequence of places are activated through the creation of tokens, as a result of the firing of transitions. The mathematical definition of a simple information flow path needs the definition of the following partial ordering:

Let $\angle$ denote the partial order as follows:
$a \angle b \Leftrightarrow$ there exists a directed path from a to b.

The mathematical definition of an simple information flow path is as follows:
A simple information flow path, x, is a set of nodes such that:
1) $\forall\ (a, b) \in x^2,\quad a \angle b$ or $b \angle a$
2) $\forall\ c \notin x,\quad \neg\,(\forall\ d \in x,\ c \angle d$ or $d \angle c)$

It should be noted that the definition of a simple information flow path corresponds to that of a line in Petri Net theory.

Definition 3.10:
In an distributed system, a *simple functionality* is an ordered sequence of processes, that operates on an input message to produce an output.

Note that other operations on other inputs may be necessary to produce the output. A simple functionality tracks the processing of a single output. A simple functionality does not provide all the necessary processes that are needed to produce an output. Nor does a simple functionality trace the only possible sequence of processes starting from a given input.

Definition 3.11:
A *complete information flow path* consists of all the simple information flow paths that end at the same sink. The mathematical definition is as follows:
A = {the set of elementary information flow paths}
$\forall\ (a, b) \in A^2,\quad a \,\mathfrak{R}\, b \Leftrightarrow$ a and b contain the same sink.

The relation $\mathfrak{R}$ is reflexive, symmetric, and transitive. Therefore, $\mathfrak{R}$ defines an order relation and there are equivalence classes.
B is a complete information flow path if and only if the three following conditions are verified:
1) B is a subset of A
2) $\forall\ (c, d) \in B^2,\quad c \,\mathfrak{R}\, d$
3) $\forall\ e \in B, \forall\ f \notin B,\quad \neg\,(e \,\mathfrak{R}\, f)$

A complete information flow path represents all the simple information flow paths that are involved in producing a specific output at some sink. A complete information flow path is a sub-net with a single sink. Therefore, in a Petri Net, there are as many complete information flow paths as the number of sinks. It should be noted that this definition does not exclude circuits from belonging to complete information flow paths. As a matter of fact, the definition of a complete information flow path does not forbid a simple information flow path to contain a loop.

Property 3.1:

In a Petri Net, a simple information flow path belongs to one and only one complete information flow path.

Proof:

Since a complete information flow path is defined by a sink, any simple information flow path included in it ends at the same sink. Therefore, it cannot belong to any other complete information flow path.

Definition 3.12:

In a distributed system, a *complete functionality* consists of the complete set of coordinated processes that operate on all the necessary inputs to produce an output.

This definition states that a complete functionality consists of the set of coordinated functions that a distributed system must activate in order to perform a task.

From the set of definitions, one can see that there is a direct correspondence between the structural properties of a system, as described by a Petri Net, and the functions that the system performs. A simple information flow path, a structural element of a Petri Net, corresponds to a simple functionality, while a complete information flow path corresponds to a complete functionality. The term functionality, in this context, is used to describe a set of coordinated functions that a system must be capable of carrying out in order to accomplish a task or sub-task. Therefore, the next step is to develop a procedure for the determination of the simple information flow paths, and for the construction of a complete information flow path.

3.4.4 Construction of the Complete Information Flow Paths

In this section, a procedure is presented to obtain analytically all the complete information flow paths in a Petri Net. In order to identify the complete information flow paths, and therefore, the elementary information flow paths that constitute each complete information flow path, the minimal support S-Invariants are used.

The methodology to obtain the complete information flow paths consists of three steps:

1) compute the minimal support S-Invariants. This step can be realized by applying the algorithm developed by Alaiwan and Toudic (1985).
2) for each minimal support S-Invariant, construct its corresponding S-component. To each minimal support S-Invariant now corresponds a sub-net.
3) By applying some rule, coalesce some of these sub-nets to obtain a complete information flow paths.

The method to create a complete information flow path from the set of S-Components of minimal support S-Invariants requires two steps. In the first step, one of the sinks of the Petri Net is chosen. The complete information flow path relative to that sink can now be constructed. First the S-Components that contain the sink are coalesced together. A complete information flow path would be obtained at that stage if all the elementary information flow paths that contain its sink were present. It is sufficient to say that the sub-net obtained is a complete information flow path if, for any node of the constructed sub-net, its number of input nodes in the sub-net equals the number of its input nodes in the original net. As the sub-net constructed previously is an S-Component, this condition is satisfied with the places already contained in the sub-net. Therefore, this condition has to be checked for the transitions of the sub-net only. This is the objective of the second step.

In the second step, the number of input places of any transition in the created sub-net is compared with the number of input places that the same transition has in the original Petri Net. If the numbers differ, that means that there is at least one simple information flow path, containing the transition, which is missing. Therefore, the missing input places are added to the sub-net. In the process, any S-Component that contains the added places is coalesced to the constructed net. In turn, these sub-nets contain other places and transitions. Therefore, the same process is done again with the transition of the newly coalesced S-Component. This recursive process, designed to add all the missing simple information flow paths, terminates when all the transitions of the sub-net have the same number of input transitions in the sub-net as in the Petri Net. Some key properties of S-Invariants are presented in Appendix B, with a classification of the S-Invariants into three categories.

Note that in the case where a place does not belong to any S-Component, that is, the net is not covered by the S-Invariants, then from this place all the directed paths from sources to that place have to be found and coalesced to the created sub-net.

This procedure is repeated for each sink of the Petri Net. The pseudocode of the algorithm to find the complete information flow paths is presented below. From the incidence matrix of the Petri Net, all the complete information flow paths are determined.

```
BEGINNING
    compute the minimal support S-Invariants   /* use of the Alaiwan and Toudic algorithm */
    For each minimal support S-Invariant Do
        construct its corresponding S-Component
    End
    identify each sink of the net               /* these are the places with no -1 in their
                                                   incidence matrix row */
    initialize the complete information flow path list   /* this list will contain S-Components */
    For each sink Do
        put all S-Component that contain the sink to the complete information flow path list
        Repeat
            initialize the newly added transitions list  /* NAT list */
            append the transitions of the S-Components to the NAT list
            For each transition of the NAT list Do
                initialize the places list          /* this list contains the missing input places   */
                                                    /* of transitions already in the created sub-net */
                If the transition does not have all its input places Then
                    Append the missing places to the place list
                End
            End
            For each place of the place list do
                append all S-Components containing the place to the complete information flow
                path list
                If there is no S-Component that contains the place Then
                    append all the paths from the sources that lead to the place to the complete
                    information flow path list
                End
            End
        Until there is no more node added to the complete information flow path list.
            /* the complete information flow path relative to the sink is constructed */
    End
END.
```

To illustrate the procedure, the complete information flow paths of the Petri Net shown in Figure 3.16 are constructed. This net contains four sources, two sinks, two information loops, and a resource loop. First the computation of the minimal support S-Invariants is done using an implementation of the Alaiwan and Toudic algorithm. There are nine minimal support S-Invariants. To each minimal support S-Invariant corresponds one S-Component. The list of minimal support S-Invariants with their S-Components is presented below:

Invariant 1: <X1> = {p1, p3, p4, p5}, and [X1] = {p1, t1, p3, t2, p4, t3, p5, t4}
Invariant 2: <X2> = {p2, p3, p4, p5}, and [X2] = {p2, t1, p3, t2, p4, t3, p5, t4}
Invariant 3: <X3> = {p6, p7, p8, p10}, and [X3] = {p6, p7, p8, p10, t5, t7, t6}
Invariant 4: <X4> = {p6, p7, p8, p11, p12}, and [X4] = {p6, p7, p8, p11, p12, t5, t7, t6, t3}
Invariant 5: <X5> = {p1, p3, p18, p19}, and [X5] = {p1, p3, p18, p19, t1, t2, t11, t12}
Invariant 6: <X6> = {p2, p3, p18, p19}, and [X6] = { p2, p3, p18, p19, t1, t2, t11, t12}
Invariant 7: <X7> = {p21, p23}, and [X7] = {p21, p23, t13, t14}
Invariant 8: <X8> = {p6, p13, p15, p14, p16, p21, p22}, and
[X8] = { p6, p13, p15, p14, p16, p21, p22, t5, t10, t8, t9, t13, t14}
Invariant 9: <X9> = {p6, p13, p15, p16, p17, p20, p21, p22}, and
[X9] = {p6, p13, p15, p16, p17, p20, p21, p22, t5, t10, t8, t9, t13, t14, t11}.

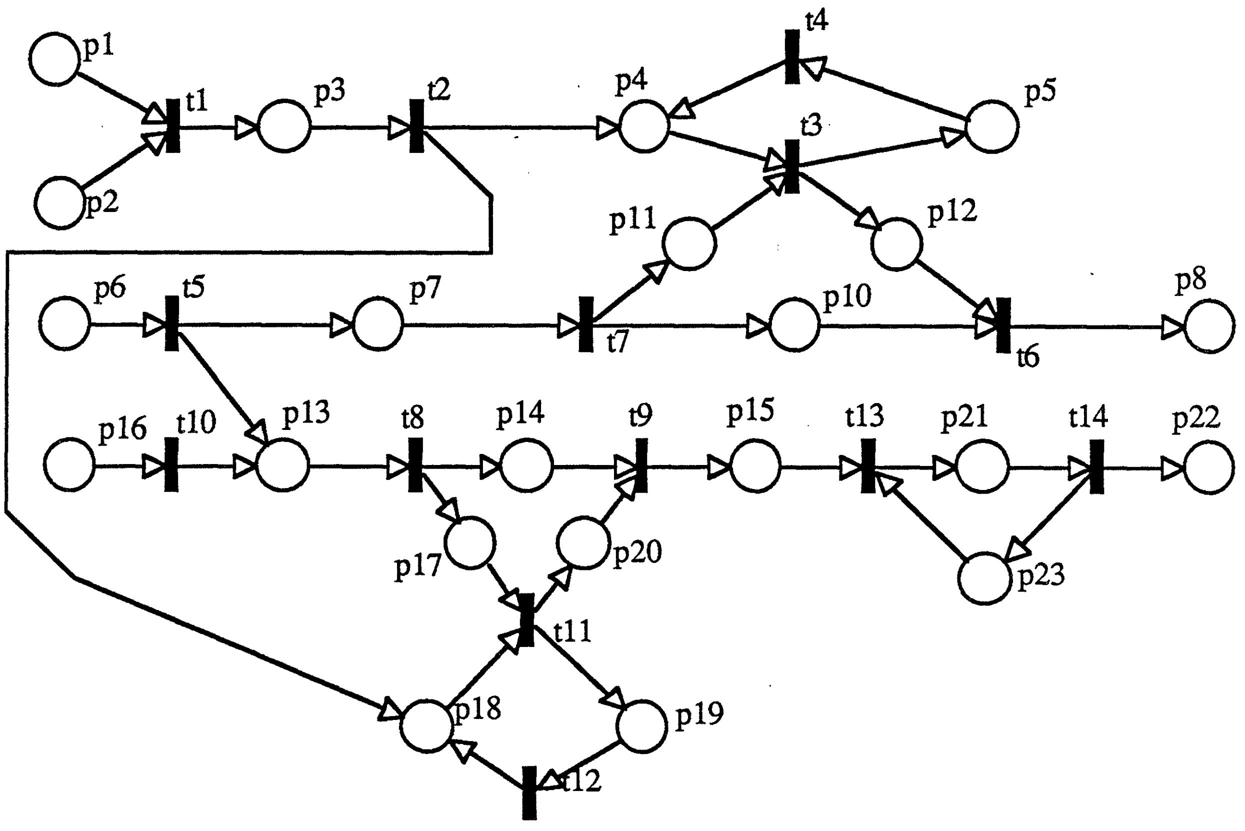


Figure 3.16 Petri Net PN7

Since the net contains two sinks, there are two complete information flow paths. The complete information flow path related to the sink p8 is constructed first. There are two

S-Components containing p8, [X3], and [X4]. Therefore the two S-Components are coalesced together. Among the transitions of [X3] and [X4], only t3 does not have all its input places in the constructed sub-net. p4 is the input place of t3 that is missing. Among the remaining S-Components, [X1] and [X2] contain p4, and are, therefore, coalesced to the previously chosen S-Components. At this stage, all the transitions of the newly coalesced S-Components have the same number of input places as in the original Petri Net. Therefore, the coalescing procedure stops, and the first complete information flow path is constructed, as shown in Figure 3.17.

The same algorithm is applied to construct the complete information flow path relative to the sink p22. There are 2 S-Components containing p22, [X8], and [X9]. Therefore, the two S-Components are coalesced together. Among the transitions of [X8] and [X9], t11 and t13 do not have all their input places in the constructed sub-net. p18 is the input place of t11 that is missing, while p23 is the input place missing for t13. Among the remaining S-Components, [X5] and [X6] contain p18, and are therefore coalesced to the previously chosen S-Components. Similarly, [X7] which contains p23 is also coalesced to the other chosen S-Components. At this stage, all the transitions of the newly coalesced S-Components, i.e. [X5], [X6], and [X7] have the same number of input places as in the original Petri Net. Therefore, the coalescing procedure stops, and the first complete information flow path is constructed, which is shown in Fig. 3.18.

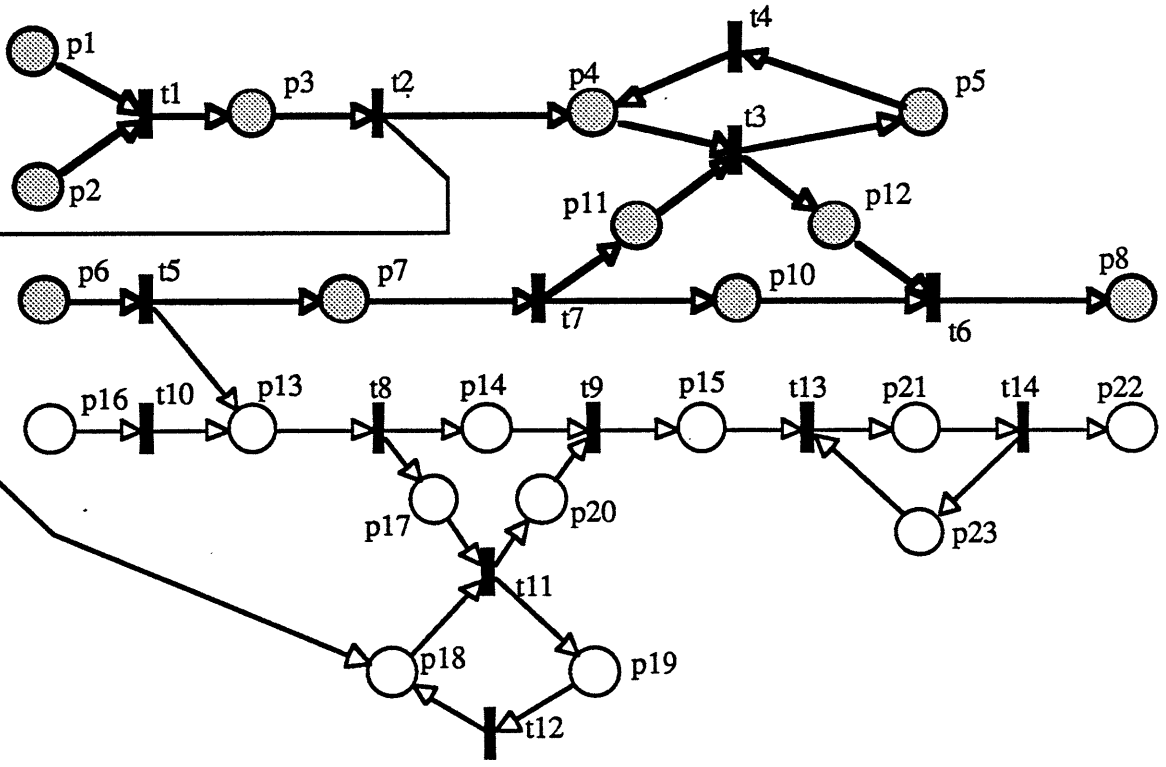


Figure 3.17 First Complete Functionality in PN7 (shown in bold face)

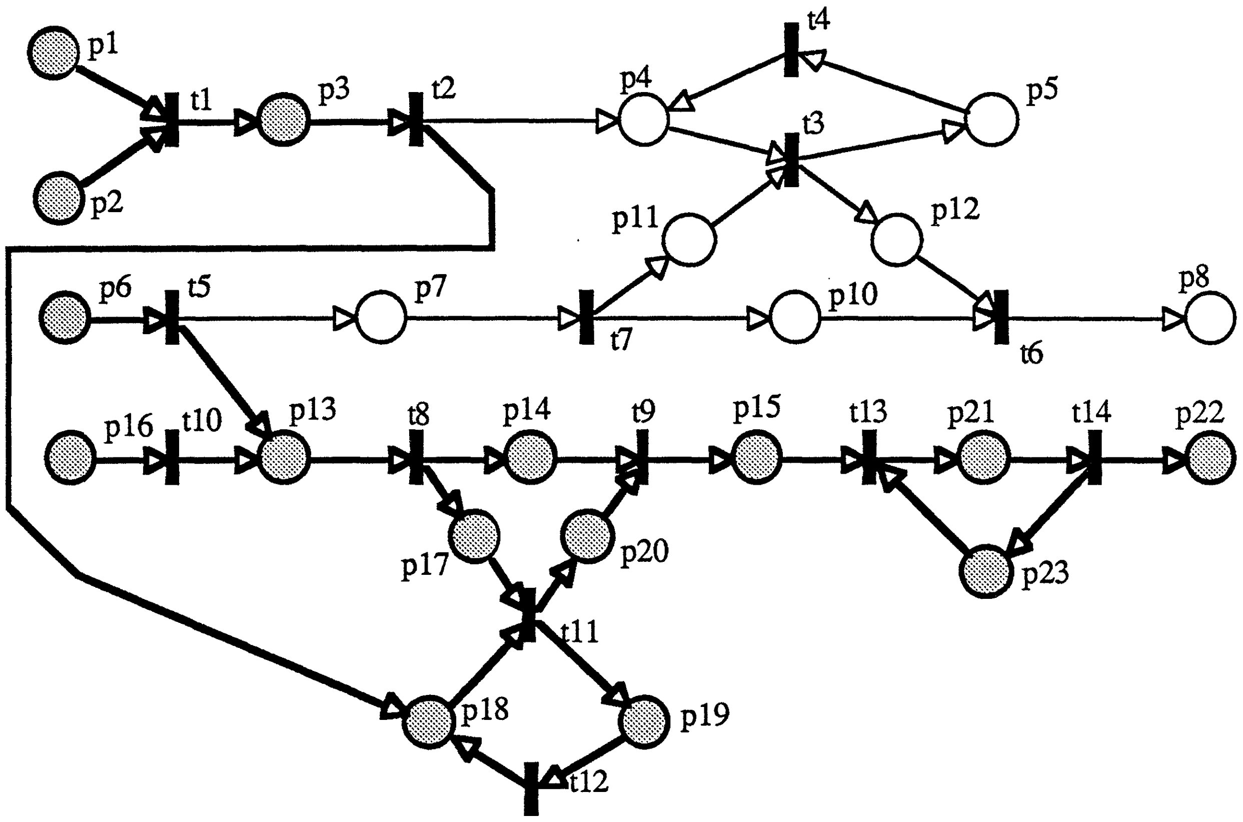


Figure 3.18 Second Complete Functionality in PN7 (shown in bold face)

3.4.5 Construction of the Simple Information Flow Paths

Once the complete information flow paths have been determined, the next problem is to compute the simple information flow paths within each complete information flow path. To do that, an enhanced version of the algorithm developed by Jin (1985) is used. The algorithm can be separated into two steps. In the first step, the incidence matrix representing the Petri Net is transformed into an interconnection matrix. The goal of this transformation is to get a simpler matrix representation of the net for the computation of simple information flow paths. An interconnection matrix indicates the connections between the transitions of the net. A link is the association of a place with one of its input connector, and one of its output connector. Therefore, if a place, pi, has *a* input connectors, i.e., *a* input transitions, and *b* output connectors, i.e., *b* output transitions, there are *a* * *b* links containing pi. In the case of sources which do not have any input connector, a link is made of the source and its output connector. Conversely, in the case of a sink, a link is made of the sink and one of its input connectors. The interconnection matrix is an m*n matrix, where m is the number of links and n the number of transitions. The elements A_{ij}, of this interconnection matrix are defined as follow:

$$a_{ij} = \begin{cases} -1 \text{ if link i leaves transition j} \\ 0 \text{ if link i is not connected to transition j} \\ 1 \text{ if link i arrives at transition j} \end{cases}$$

The matrix is constructed row by row by considering each link that leaves a transition (-1) and indicating the transition to which it is directed (+1). Since each row represents a single link, it must have exactly one +1 and one -1 and, consequently, the sum of its elements must be zero. This is not the case, of course, for the links that connect sources to transitions and the links that connect transitions to sinks. The former are recognized by a row with a single +1, while the latter by a row with a single -1.

In the second step, the interconnection matrix is used to compute the simple information flow paths. The problem is to find, for each source of the complete information flow path, all possible simple information flow paths from that source to the sink. The transition that follows immediately the source place becomes the main root of the tree that represent the simple information flow paths; every simple information flow path forms a branch of the tree.

The elements of the interconnection matrix, a_{ij}, can be interpreted as follows:

- if $a_{ij} = -1$ and $a_{ik} = 1$, then transition tj precedes and is connected to transition tk.
- if there are more than one (-1) in a column j, then transition tj is a root or a sub-root of the tree.
- if there are *q* (+1) in a column j, then *q* simple information flow paths coalesce after they reach transition tj.

With these properties in mind, the matrix is scanned to determine the simple information flow paths. The scanning starts by identifying rows that have a single non-zero entry, a -1. This corresponds to a link that terminates at a sink. Say that the -1 occurs in column i. The first (+1) in column i of A is the first input to ti and it is processed first. The scanning stops when a multi-input transition tj is encountered, i.e., a transition tj on which several paths converge. The transition tj is marked as a sub-root, i.e., the end of some sub-path. After all inputs to the original transition ti are processed, the scanning restarts to determine if there is another row with a single non-zero element equal to -1 as the ith column. The same procedure is applied to all the stored sub-roots. When the sub-paths of the last sub-root end at t1, which is the first transition linked to the source, scanning is complete. In this scanning process, loops are identified as being sub-paths where the first transition in the sequence is the same as the last transition.

Let W(k, m) be the mth sub-path ending at transition tk. After all sub-paths are found, paths are constructed by concatenating sub-paths. This is accomplished by matching the last transition ti in sub-path W(k, m) to the first transition ti in sub-path W(i, m'). When the last transition is t1, a simple information flow path has been completed for the set of input sources related to t1.

To illustrate the mechanics of the algorithm, the simple information flow paths of the complete information flow path related to sink p22 in Figure 3.16 are computed. The algorithm starts by constructing the interconnection matrix from the incidence matrix representing the complete information flow path. This incidence matrix is obtained from the incidence matrix of the net by deleting the rows corresponding to the places of the net that do not belong to the complete information flow path, and the columns corresponding to the transitions of the net that do not belong to the complete information flow path. From the interconnection matrix, ten sub-paths are found. These are:

sub-path 1: t13→t14
sub-path 2: t13→t14→t13
sub-path 3: t9→t13
sub-path 4: t11→t12→t11
sub-path 5: t1→t2→t11
sub-path 6: t8→t11
sub-path 7: t11→t9
sub-path 8: t8→t9
sub-path 9: t10→t8
sub-path 10: t5→t8

From the sub-paths, the simple information flow paths are constructed. Starting from the output transition of the sources to the input transition of the sinks, there are sixteen simple information flow paths. These are:

t1→t2→t11→t12→t11→t9→t13→t14→t13→t14
t1→t2→t11→t12→t11→t9→t13→t14
t1→t2→t11→t9→t13→t14→t13→t14
t1→t2→t11→t9→t13→t14
t10→t8→t11→t12→t11→t9→t13→t14→t13→t14
t10→t8→t11→t12→t11→t9→t13→t14
t10→t8→t11→t9→t13→t14→t13→t14

t10→t8→t11→t9→t13→t14

t10→t8→t9→t13→t14→t13→t14

t10→t8→t9→t13→t14

t5→t8→t11→t12→t11→t9→t13→t14→t13→t14

t5→t8→t11→t12→t11→t9→t13→t14

t5→t8 →t11→t9→t13→t14→t13→t14

t5→t8→t11→t9→t13→t14

t5→t8→t9→t13→t14→t13→t14

t5→t8→t9→t13→t14

These paths are complete for the three sets of sources related to t1, t5, and t10, respectively. t1 has two input sources, p1 and p2, t5 has one input source, p6, and t10 has one input source, p16. Therefore, there are four simple information paths originating in p1, four in p2, six simple information flow paths originating in p6, and six simple information flow paths originating in p16. Finally, there are twenty simple information flow paths within the complete information flow path.

3.5 SUMMARY

In this chapter, after describing the Petri Net models of both the requirements and the design of a distributed system, the relation between structural properties of the Petri Net model of a distributed system and its functionality have been established. The sub-nets of interest have been identified, and the procedures to construct these sub-nets have been explained. From now on, the goal is to proceed to the evaluation of the functionality of a proposed or implemented system versus its requirements.

CHAPTER IV

EVALUATION OF FUNCTIONALITY IN DISTRIBUTED SYSTEMS

4.1 INTRODUCTION

In this chapter, the functional evaluation of a proposed or implemented system against its requirements is presented. At this point, there are two Petri Nets, one describing the required functionality, and one describing the operation of the proposed or implemented system. Throughout this chapter, the first Petri Net will be called the *Requirements Net*, while the latter will be called the *System Net*. These two Petri Nets are specific to a particular scenario. As discussed in the previous chapter, when a particular scenario is selected, both Petri Nets are modified by eliminating functions and unnecessary sources and sinks. Therefore, both Petri Nets are reviewed and reduced to reflect the chosen scenario. Each Petri Net may contain switches that reflect different possible strategies. Also, the existence of redundant functions may be translated into the existence of several occurrences of the same strategy.

The next task in the evaluation process is to remove the switches. By removing the switches, a set of ordinary Petri Nets is obtained, for each of the two Petri Nets. As of now, an ordinary Petri Net inferred from the Requirements Net will be called a *Required Strategy Net*, while an ordinary Petri Net inferred from the System Net will be called a *System Strategy Net*. Each ordinary Petri Net corresponds to the implementation of a strategy. On the other hand, a particular strategy may be represented by one or more different ordinary Petri Nets.

Therefore, the next task is to sort the Petri Nets to recognize which Required Strategy Net corresponds to which System Strategy Net, all implementing the same strategy. This means that the evaluation of functionality must be done, in the case of a specific scenario, on a strategy by strategy basis.

For a given scenario, there may be *p* Required Strategy Nets, $p \geq 1$, and *q* System Strategy Nets, $q \geq 1$ corresponding to the same strategy. In this case, the functional evaluation will be done between each Required Strategy Net and each System Strategy Net, i.e., there will be $p \times q$ evaluations performed in the case of that particular strategy.

This whole decomposition process is depicted in Figure 4.1. From the two Petri Nets representing the model of the proposed or implemented system, and of the requirements, the selection of a scenario yield the System Net and the Required Net. The removal of the switches in the system net produces the set of System Strategy Net, while the same process applied to the Required Net produces the set of required Strategy Nets.

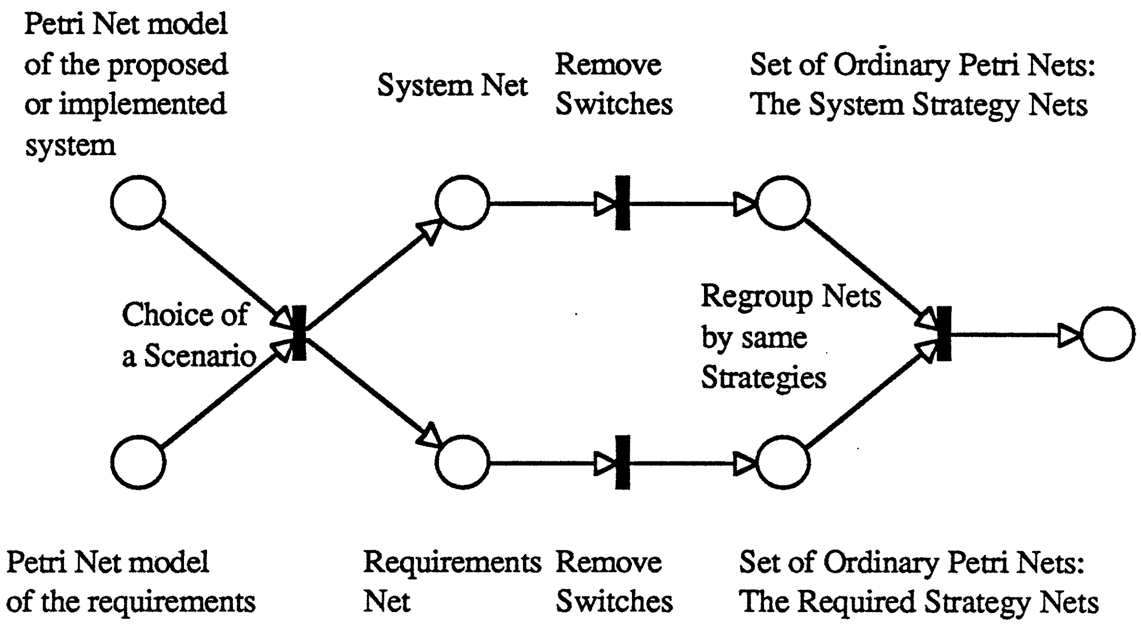

Petri Net model of the proposed or implemented system
System Net
Remove Switches
Set of Ordinary Petri Nets: The System Strategy Nets
Choice of a Scenario
Regroup Nets by same Strategies
Petri Net model of the requirements
Requirements Net
Remove Switches
Set of Ordinary Petri Nets: The Required Strategy Nets

Figure 4.1 The Decomposition Process

Table 4.1 shows an example of the arrangement of Petri Nets in the representation of the requirements and the system. Suppose, for a given scenario, that there are five different Required Strategy Nets, each one corresponding to a different strategy, A to E, and that there are *a* System Strategy Nets corresponding to the strategy A, *b* System Strategy Nets corresponding to the strategy B, and so forth. Therefore, for each strategy, the evaluation is performed between the Required Strategy Net and each System Strategy Net. There are $1 \times a$ evaluations to be performed in the case of strategy A.

The problem is to identify strategy nets among the set of ordinary nets inferred from the System Net that correspond to a particular strategy in the Required System Net. This is done by identifying the particularity of a strategy in a Required Strategy Net that is also present in a System Strategy Net, such as the existence of a particular function, or a particular ordering of functions. This can also be a set of inputs/outputs.

Table 4.1 Strategy correspondence between Petri Nets

Strategy	Required Strategy Net	System Strategy Net
A	1	a
B	1	b
C	1	c
D	1	d
E	1	e

Ultimately, several pairs of two ordinary Petri Nets are identified for comparison, one from the Requirements Net, and one for the System Net. For each of the two nets in a pair, the complete information flow paths and the simple information flow paths are constructed. Shortfalls and overlaps can now be determined by comparing the structure of the two Petri Nets. Underlying the whole issue of comparison is the notion of net equivalence and, specifically, functional equivalence. It is not expected that the two sub-nets will be identical; what is of interest is whether the information receives functionally equivalent processing and that the same information, processed in a comparable manner, is used to produce the outputs. This issue becomes even more important in the consideration of overlaps, although here, in some cases, the theory of Petri Nets can be used to distinguish between redundancy that improves the survivability of the system and redundancy that causes conflict or confusion.

4.2 SHORTFALLS

The identification of shortfalls and overlaps is based on the comparison of two sets of complete information flow paths and their embedded simple information flow paths. However, in both situations, whether shortfalls or overlaps, several cases can be identified.

4.2.1 Complete Shortfall

The most extreme case is of the absence of a complete functionality from the System Strategy Net which is present in the Required Strategy Net. That corresponds to the absence of a sink in the System Strategy Net. Formally, given the correspondence between complete functionality and complete information flow path, this type of shortfall is defined as follows:

Definition 4.1:

A *complete shortfall* is observed if a complete functionality in the Required Strategy Net has no counterpart in the System Strategy Net, i.e., if a complete information flow path in the Required Strategy Net has no corresponding subnet in the System Strategy Net.

The most obvious test is to compare the sink nodes of the two nets. Since a complete functionality has been associated with each sink of a net, the first step is to list the sinks of the two nets and to establish their correspondence. Note that the correspondence need not be one to one. It is possible that two or more sinks in the Required Strategy Net correspond to a single sink, and vice-versa. Therefore, when this is done, the test is to establish correspondences between the sinks in the Required Strategy Net and in the System Strategy Net. If a sink in the Required Strategy Net does not have any counterpart in the System Strategy Net, then there is a complete shortfall.

For example, suppose that the Required Strategy Net has five sinks, Si1, Si2, Si3, Si4, and Si5, and that the System Strategy Net has four sinks, si1, si2, si3, and si4. If the output produced by Si1 is equivalent to the combination of the outputs produced by si1 and si2, if the combination of the outputs produced by Si2 and Si3 are equivalent to the output produced by si3, and if the output produced by Si4 is equivalent to the output produced by si4, then clearly, there is no counterpart to the output produced by Si5. These equivalence relations between sinks are summarized in Figure 4.2, where correspondences, represented by lines, are made between the set of sinks in the Required Strategy Net and in the System Strategy Net. The vertical bars joining lines represent the logical operator AND. Therefore, there is a complete shortfall in that System Strategy; it is not capable of producing an output corresponding to Si5.

4.2.2 Partial Shortfall

If there exists a one to one correspondence between the set of sinks in the Required Strategy Net and the set of sinks in the System Strategy Net then the next step is to compare each set of complete information flow paths attached to a set of sinks in the Required Strategy Net with the set of complete information flow paths attached to the set of corresponding sinks in the System Strategy Net. This comparison is to determine if the sub-net that corresponds to a required complete functionality is contained in the sub-net of the corresponding complete functionality in the System Strategy Net.

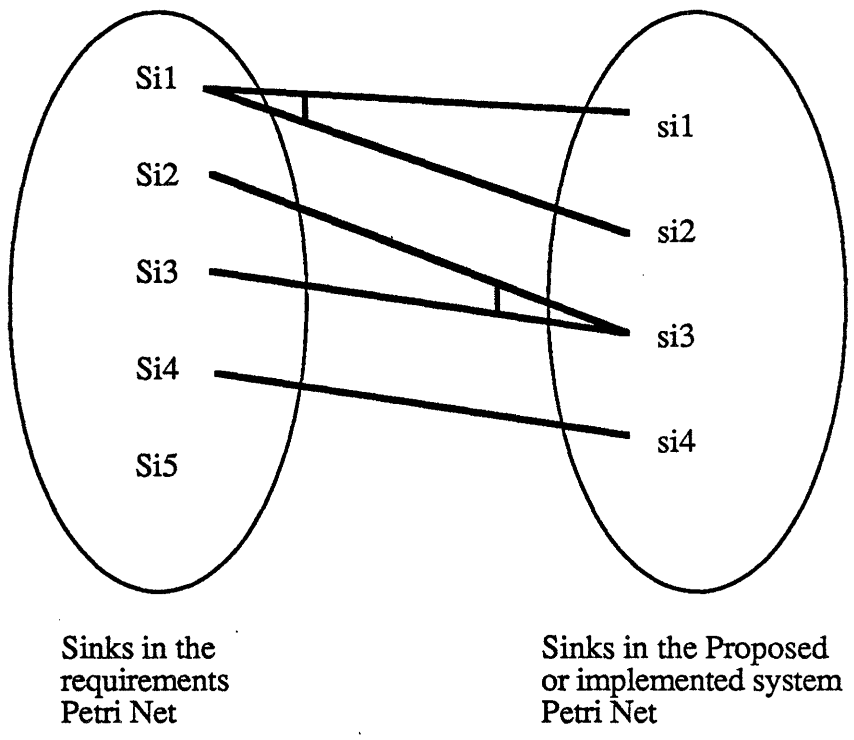

Si1
Si2
Si3
Si4
Si5
si1
si2
si3
si4
Sinks in the requirements Petri Net
Sinks in the Proposed or implemented system Petri Net

Figure 4.2 Example of Complete Shortfall

Definition 4.2:

A *partial shortfall* is observed when the complete information flow path in the System Strategy Net that corresponds to a required complete functionality does not contain all the simple information flow paths corresponding to all the embedded simple functionality in the required complete functionality.

Partial shortfall addresses the issue of diminished functionality existing in the system, e.g., an output is produced either with reduced processing (some processing steps are missing), or with reduced inputs (some data are either not available or not used). The latter case is easier to check because it relates directly to the sources and the simple information flow paths.

4.2.3 Detection of Partial Shortfalls

To detect partial shortfalls, the simple information flow paths of corresponding complete functionalities in the Required Strategy Net and in the System Strategy Net have to be compared. For example, in the case depicted in Figure 4.2, the simple functionalities of the complete functionality associated with si1 and si2 should be compared with the simple functionalities of the complete functionality associated with Si1, and so forth. Such a comparison requires some knowledge about the sources and functions in both nets, and more specifically, the establishment of correspondence between sources, and functions embedded in both Petri Nets. Therefore, for each subnet, some explicit labeling is needed in order to:

- distinguish between transitions representing functions or processes, and other transitions representing either transmission of information, request for information, acknowledgment messages, protocols, and artifacts of the model (such as nodes artificially added to prevent self-loops).
- distinguish between the multiple occurrences of a function e.g., F1 in workstation W1 and F1 in workstation W2.
- eventually take into account the decomposition of a function into a sequence of subfunctions.

The proposed labeling is as follows:

- any transition representing a function starts its name with an f, the others do not.
- The general format of a transition representing a function is fx-y-z, where x, y, and z are integers: x represents the type of function, y accounts for a specific subfunction of fx, i.e., if fx is decomposed into a sequence of two subfunctions, then fx-1 and fx-2 result. Lastly, z accounts for the occurrence of the function, i.e., if there are two equivalent fx in the system, then there is fx-1-1 for the first one, and fx-1-2 for the second one.

With such a labeling, it is possible to establish correspondences between equivalent functions within each net. An example is shown in Figure 4.3. There are two simple information flow paths. One of these contains function f1-1 which is decomposed into the sequence f1-1-1 → f1-2-1. The other contains f1-2 which is equivalent to f1-1 (both have the same first number in the labeling). f1-2 is decomposed into the sequence f1-1-2 → f1-2-2 → f1-3-2.

Once the equivalence between functions is achieved, the procedure to identify partial shortfalls is as follows:

Step 1:

For each set of complete information flow paths, eliminate in the sequence of transitions describing each simple information flow path the transitions not representing processes (this is done easily by scanning the labels). As a matter of fact, only sequences of processes are of interest for a functional analysis (this may not be the case when an evaluation of performance is undertaken).

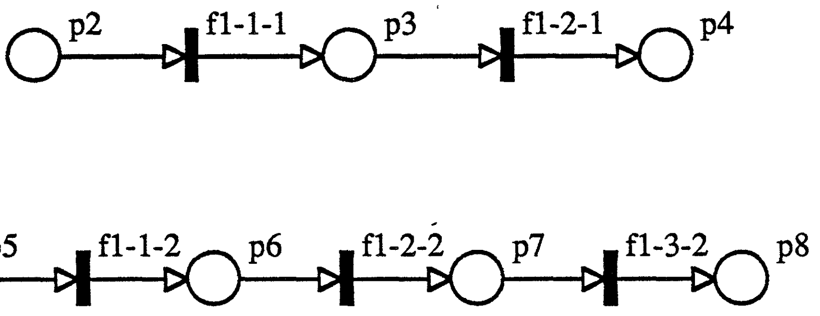


Figure 4.3 Two equivalent Simple Information Flow Paths

Step 2:

For each set of complete information flow paths, it may happen that, as a result of Step 1, some paths become identical, i.e. start from the same source, and contain *the same sequence of the same functions*. Therefore, the number of paths is somewhat reduced. A simple information flow path is then characterized by an input source and a sequence of functions.

Given a simple information flow path in the Required Strategy Net, then a search is undertaken for finding equivalent simple information flow paths among the set of the System Strategy' simple information flow paths.

Step 3:

Scan the sequence of functions embedded in the Required Strategy' simple information flow path. Because of the correspondence already established between the functions in the Required Strategy Net and in the System Strategy Net, simple information flow paths with sequences of functions which match with those of the requirements' simple information flow paths are identified.

Rule 1: If a simple functionality, represented by a simple information flow path in the Required Strategy Net, has no equivalent in the System Strategy Net, then there is a partial shortfall.

If there are such paths, then the next step is to match the paths from the Required Strategy

Net with the paths of the System Strategy Net, in accordance with the correspondence between sources.

An example is shown in Figure 4.4. Suppose that the upper Petri Net, PNU, is a simple information flow path for the Required Strategy Net. It is made of the functional sequence f1-1 → f2-1. In the System Strategy Net, the lower Petri Net, PNL, there are three simple information flow paths, t1 → F1-1 → F2-1, t2 → F1-1 → F2-1, and t3 → F1-1 → F2-1. If F1 is equivalent to f1, and F2 to f2, then there are three simple information flow paths that correspond to PNU because transitions t1, t2, and t3 do not represent a process. However, the source p1 encompasses three classes of input signal, namely c1, c2, c3, which are the sources of each simple functionality in PNL. Therefore, in this case, the simple information flow path in the requirements is functionally equivalent to the combination of the three simple functionalities. In fact, the structure of the proposed or implemented system Petri Net is such that p7 can be considered as a source that is equivalent to p1.

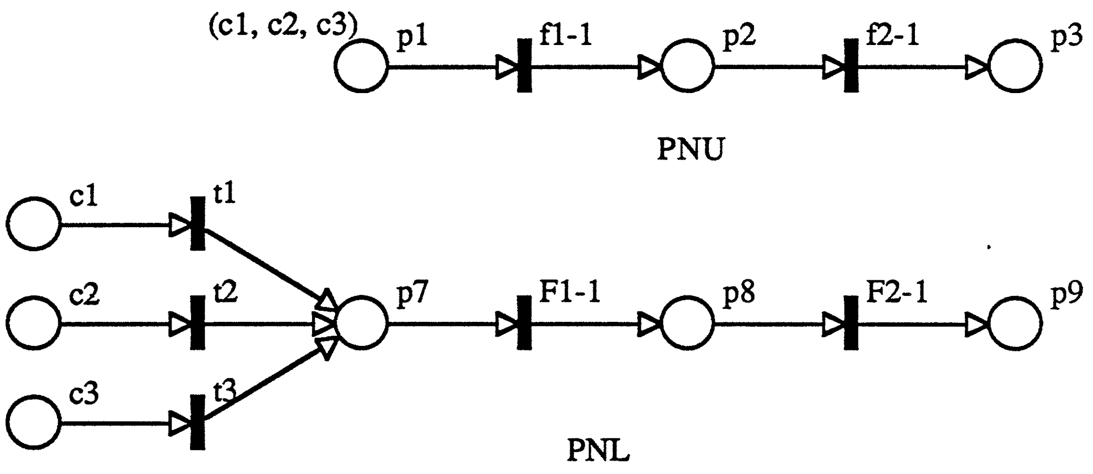


Figure 4.4 Two functionally equivalent sub-nets

Rule 2: If the sources of a simple functionality specified in the Required Strategy Net are not contained in the sources of the simple information flow paths of the System Strategy Net that have an equivalent sequence of processes, then there is no functional equivalence.

Whether or not a required source for a simple functionality that is not present in the system net constitutes a shortfall is a difficult question to answer by inspecting the mere structure of the net. As a matter of fact, that discrepancy may result from a difference in the level of description of the two nets. It may also be possible that the requirements impose unnecessary processing of some input by some simple functionality that has been corrected by the designer.

4.3 OVERLAPS

The Third Webster International dictionary defines an overlap as:

- To have something in common with, to comprehend elements of, to coincide in part with.

In the context of distributed systems, an overlap means that a certain sequence of operations on a given input can occur in more than one way in the net. For example, signals from a source can go to different workstations and be processed in parallel to reproduce the same output signal. If this capability is used in the sense of alternative means of processing the same information, i.e., only one of the paths is used for each individual message that is processed, then this type of redundancy is beneficial because it increases the survivability of the functions in a distributed system. However, explicit protocols must be in place that determines without any ambiguity the selection of the particular single path to process the signal. If the protocols are not well designed, a conflict may occur which results in confusion. Consider now the case where a given input signal is processed in parallel by several paths. If only one of the outputs is used and the results of the other paths are ignored, then this parallel processing represents a waste of processing resources. The question is then to evaluate if this waste is prejudicial to the good functioning of the system. If the workstations carrying the parallel processes could have been carrying instead other functions, thus reducing the time of the whole task, then clearly the answer is yes. If, on the other hand, the workstation could not have processed any other function that is, at the stage of the processing, critical to continue the whole processing, then the answer is no.

On the other hand, if the output of all the paths are fused together prior to the final output being processed, then the quality of the output may improve at the cost of increased delay and increased coordination. A trade-off is identified between increased accuracy and shorter processing time. Here again, it may be more effective to dedicate one of the workstations performing one of the parallel equivalent processes to the processing of another concurrent function so as to reduce the processing time of the whole system.

4.3.1 Redundancy with Conflict

Consider, for example the functionality represented by the simple information flow path shown in Figure 4.5. The signal undergoes two processes, represented by transitions t1 and t2, from the source p1 to the sink p3. Assume that this represents a desired functionality embedded in the requirements net.

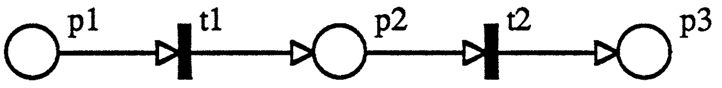


Figure 4.5 A Simple Functionality

Now consider the net shown in Figure 4.6. The source is p1, the sink is p3, but there are two alternative simple information flow paths leading from p1 to p3. The top path is identical to that of Figure 4.5. Let us assume that the bottom path is equivalent to the top path, i.e., the two processes represented by t3 and t4 produce the same result as the processes t1 and t2, even though the intermediate signal in p2 and p4 may be different. This Petri Net represents an overlap as the two simple functionalities are equivalent. However, this is not a desirable redundancy because it creates conflict; there is no rule associated with p1 to determine which of the two transitions, t1 or t3, will fire or execute when a token appears in p1. This kind of redundancy is detrimental to the efficient operation of the system, and reflects weakness in the concept of operations.

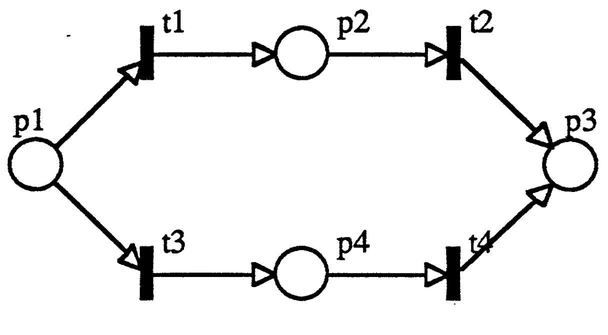


Figure 4.6 Redundancy with Conflict

The solution to that problem, as already described in the previous chapter, is to implement a switch, s1. This is shown in Figure 4.7. The meaning of the switch is that only one of the places p5 and p6 will receive the signal, i.e., the token that appears in p1. The choice will be made according to the rule that is embedded in the switch. Confusion is avoided by forcing the clarification of the concept of operation. This type of structural change in the description of the proposed or implemented system net is to be made prior to the identification of pairs of Petri Nets to be compared, as the introduction of a switch creates new strategies.

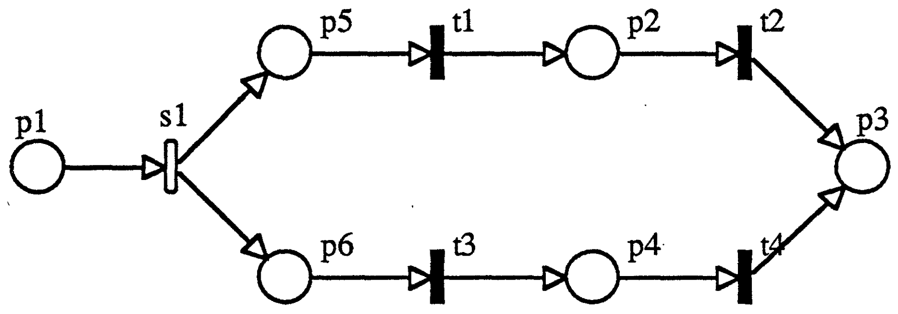


Figure 4.7 Redundancy with Resolved Conflict

4.3.2 Redundancy with Concurrency

Since it has been assumed that the Petri Nets obtained in this manner are decision free, there are two types of redundancies left. The first type is depicted in Figure 4.8. In this case, the incoming signal goes to both simple information flow paths and the two outputs arrive at the sink independently. While more resources are used, both speed of response and higher survivability may be achieved because the sink can receive either one of the processed signals and the rest of the system can proceed as soon as the first one arrives.

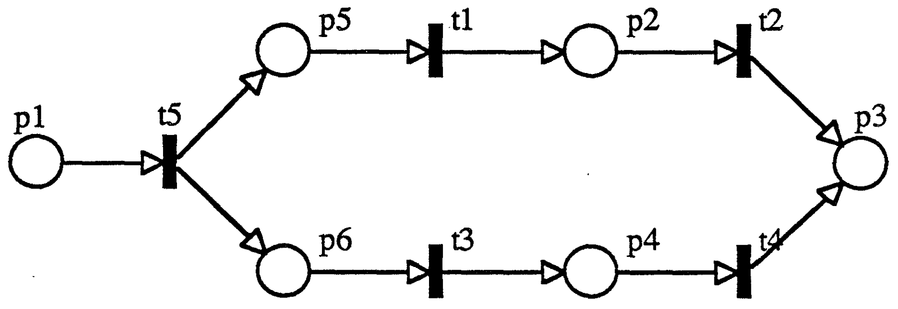


Figure 4.8 Redundancy with Concurrency

4.3.3 Redundancy with Synchronization

The opposite is true in the redundancy illustrated in Figure 4.9. As in the case of redundancy with concurrency, both simple information flow paths process the incoming signal, but the sink can accept only a fused result. This need for fusion is modeled by transition t6. Both simple information flow paths must complete their execution in order for t6 to be enabled, so that it can fire and produce the output. Clearly, in this case, the response time will be the maximum of the response time of the two processing paths, while in the case of redundancy with concurrency, the response time is the shorter of the two processing times. On the other hand, the quality of the output may be enhanced, provided the fusion algorithm is adequate.

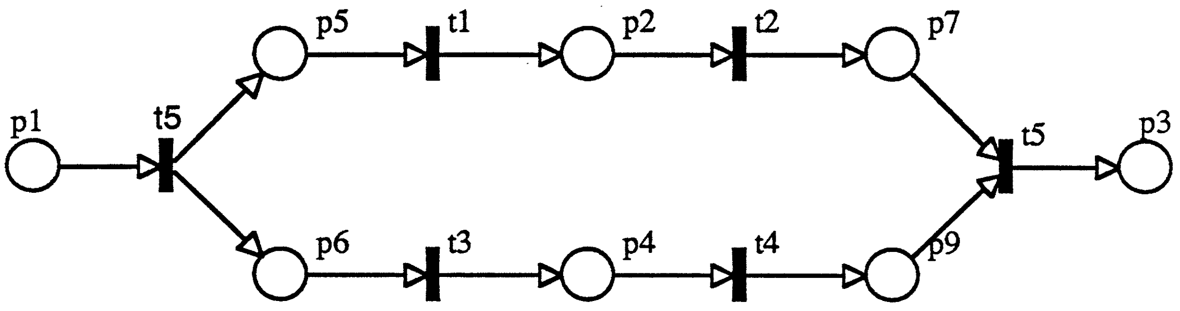


Figure 4.9 Redundancy with Synchronization

Thanks to the construction of the complete information flow paths and of the simple information flow paths, the Petri Nets representing the Required Strategy and the System Strategy can be decomposed into structural blocks. The analysis of these blocks leads to the identification of overlaps and their classification according to the presence of conflicts and to the presence of concurrency or synchronization.

4.3.4 Determination of Redundancies

Definition 4.3:

Two simple information flow paths belonging to the same complete information flow path are *redundant* if and only if the following two conditions are fulfilled:

- i) the two simple information flow paths have the same source (and sink)
- ii) the two simple information flow paths have the same sequence of functions, i.e, the same sequence of equivalent occurrences of functions.

In short, two simple functionalities are redundant, if they represent the same sequence of operations on a given input.

By inspecting the sequence of transitions of two equivalent simple information flow paths, the last common transition before both simple information flow paths diverge is determined. From that transition, a new common transition is searched. If there is no such transition, then this means that the two simple information flow paths coalesce at the sink of the complete information flow path. Therefore, the two paths exhibit a redundancy with concurrency. On the other hand, if such transition exists, then the two simple information flow paths exhibit redundancy with synchronization.

4.4 COORDINATION

4.4.1 Definition

Lastly, the issue of coordination of simple functionalities has to be addressed. It is recalled that a functionality has been defined as a set of coordinated functions. Therefore, in the case where no shortfall has been evidenced, the question arises as to whether or not the coordination between the simple functionalities in the proposed or implemented system is similar to that of the requirements.

Definition 4.4:

Two simple functionalities are *coordinated* if their corresponding simple information flow paths coalesce at some transition.

This test is designed to evaluate if, for two required coordinated simple functionalities there exist two equivalent simple functionalities in the System Strategy Net coordinated the same way. Figure 4.10 and 4.11 shows an example. Suppose that in the Required Strategy Net, there are two simple information flow paths, Pa1 and Pa2 which are respectively:

- Pa1: $c1 \mid F1 \rightarrow F3$
- Pa2: $c2 \mid F2 \rightarrow F3$

Suppose that in the System Strategy Net, there are two simple information flow paths, pa1 and pa2 which are respectively:

- pa1: c'1 | f1 → f3-1
- pa2: c'2 | f2 → f3-2.

Suppose f1 is equivalent to F1, f2 to F2, f3-1 and f3-2 to F3, and that source c1 is equivalent to c'1, and source c2 is equivalent to c'2. Then Pa1 is equivalent to pa1, and Pa2 is equivalent to pa2.

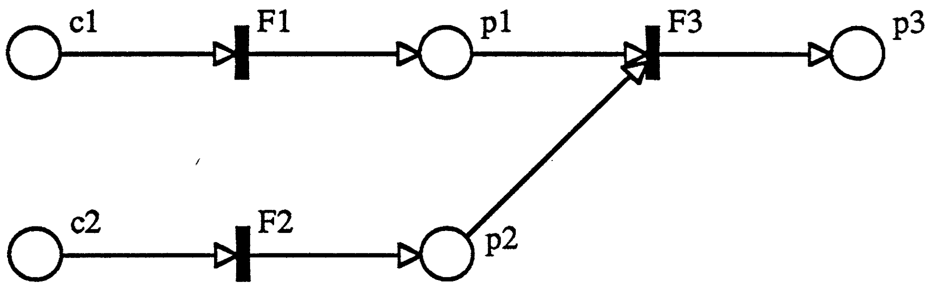


Figure 4.10 Petri Net PN8

However, Pa1 and Pa2 are coordinated at F3. On the other hand, neither of the two transitions corresponding to F3 in the implemented or proposed system net, i.e. f3-1 and f3-2, require coordination. Therefore, the coordination between the two sets of equivalent simple functionalities is not the same.

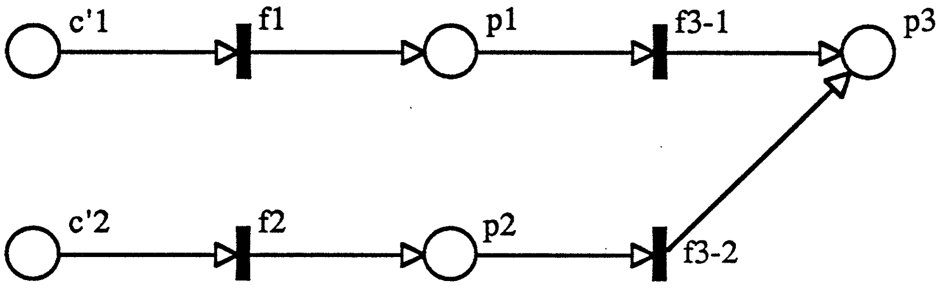


Figure 4.11 Petri Net PN9

Now, suppose that instead the system net looks like the net shown in Figure 4.12.

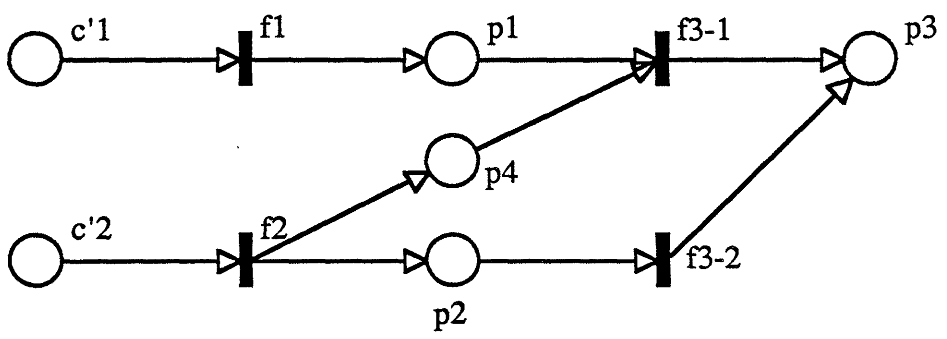


Figure 4.12 Petri Net PN10

In this case there are three simple information flow paths, pa1, pa2, and pa3 which are respectively:

- pa1: c'1 | f1 → f3-1
- pa2: c'2 | f2 → f3-1
- pa3: c'2 | f2 → f3-2.

Suppose, as in the previous example, that f1 is equivalent to F1, f2 to F2, f3-1 and f3-2 to F3, and that source c1 is equivalent to c'1, and source c2 is equivalent to c'2. It is then clear that pa1 is equivalent to Pa1, and that pa2 and pa3 are equivalent to Pa2. Pa1 and Pa2 are coordinated at F3. Similarly, pa1 and pa2 are coordinated at f3-1, which is equivalent to F3. Therefore, pa1 and pa2 are equivalently coordinated with Pa1 and Pa2. On the other hand, f3-2, which is equivalent to F3, is contained only in Pa3. Therefore, the coordination is not the same as in the requirement net for f3-2.

Such problems of coordination between the system net and the requirements net are due to one of the three following reasons:

- the existence of a partial shortfall. In this case, the lack of a simple functionality shows in the list of simple functionalities using a particular process.
- A design flaw in the system, such that effectively a simple functionality is missing for a given process. This case is different from the previous one in the sense that they may not be a partial shortfall, but the number of equivalent simple functionalities is less than the number of occurrences of the same function. This is the case in Figure 4.12, where there is no partial shortfall, but there are two functions f3, and only one simple functionality of type f2 → f3.
- A deadlock resulting from the mere implementation of a particular strategy, a particular

switch setting in the Petri Net. This is translated at the function level by the lack of the arrival of a token, resulting in the incapacity of the function to produce its output. This case is illustrated in Figure 3.10 which shows a net with a switch, s1. Note that, in this case, whatever the setting, there is always a deadlock, as transition t3 has only one input place with a token, either p4 or p5, instead of tokens in both input places.

4.4.2 Determination of Coordination Problems

To determine coordination differences between simple functionalities in the two Petri Nets, the approach is to relate the equivalent transitions, i.e. the processes, that belong to more than one simple information flow path. Therefore, for each transition representing a function of the Required Strategy Net, the list of simple information flow paths containing that transition is constructed. The same is done for the transitions of the System Strategy Net.

Given a function in the Required Strategy Net, this function is contained in the sequence of functions of one or more simple information flow paths. The equivalent functions in the System Strategy Net are then checked to verify that they, too, are contained in the sequence of functions of equivalent simple functionalities. If this is so, then there is no coordination problem. If this is not so, then there is a coordination problem.

If there is a coordination problem, the next step is to determine the nature of the coordination problem, whether it leads to deadlock or not. To do that, it is necessary to compare the number of input places of the transition representing the function in the strategy net with the number of input places that the same transition has in the original net with switches. If these two numbers differ, then there is deadlock, while if they match, there is no deadlock.

4.5 EVALUATION OF FUNCTIONALITY

When the evaluation of pairs of ordinary nets is completed, it is then possible to evaluate the set of alternative structures in order to assess how well the proposed or implemented system meets its requirements. The goal of this evaluation is to identify functional problems in the design of the system.

It is clear that a particular ordinary Petri Net in the proposed or implemented system net which exhibits either a complete or a partial shortfall does not operate according to the requirements. The switch setting corresponding to that Petri Net does not allow the implementation of the functional requirements. The implementation of that structure of the system net is not valid and the corresponding switch setting is to be discarded. One case of interest is when all the ordinary Petri Nets corresponding to a particular strategy contain the same partial shortfall. In this case, it is possible that this strategy cannot be implemented satisfactorily. A way has to be found to implement the missing simple functionality within the existing structure of the net. This is sometimes possible, if all the functions contained in the missing simple functionality have corresponding functions in the System Strategy Net.

The lack of some sources in the processing of a simple functionality may not be critical. The operation of that simple functionality and its contribution to the mission or goal of the system has to be evaluated in greater detail so as to decide whether or not there is an operational problem. It may happen that some data considered critical for a simple functionality in the requirements are in fact not essential for the fulfillment of the mission.

The issue of redundancy is more sensitive. Clearly, a redundancy with conflict is detrimental to the functional effectiveness of the system. Further, the uncertainty that results is also a factor increasing the chance of deadlock. A redundancy with concurrency is a good thing as it increases survivability and reliability. A redundancy with synchronization is somewhat dangerous in the sense that reliability decreases compared to an equivalent processing structure with no redundancy. However, the fusion algorithm associated with such a redundancy may lead to improvement in the quality of the output. Therefore, there is a trade-off to be evaluated on a case by case basis. The most important thing is to be able to detect such a redundancy, so that an evaluation of the trade-off can be performed.

There can be three reasons for lack of coordination at a given function; (a) a bad switch setting that removes some simple information flow paths, thus resulting in deadlock, (b) the existence of a partial shortfall, or (c) a missing simple functionality containing that function although there is no partial shortfall, i.e., the number of redundant function is greater than the number of redundant simple functionalities. A lack of coordination at some function has to be evaluated at a global level. Given the distribution of the processing and some possible redundancy, problems of lack of coordination between simple functionalities may not be critical. It depends on the nature of the coordination problem which may be inferred from the structural arrangements of the different possible Strategy Petri Nets.

For example, in the case of the net shown in Figure 4.12, if the lack of coordination of function f3-2 results in deadlock, then the system will still accomplish its mission, through function f3-1. This is the case if that particular structure is inferred from the Petri Net with switches shown in Figure 4.13.

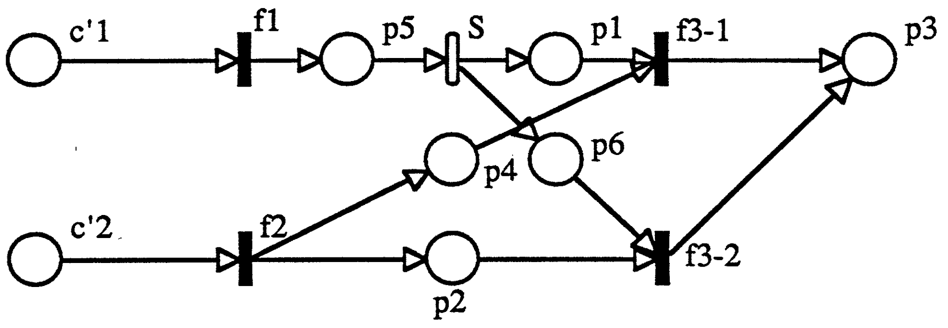


Figure 4.13 Petri Net with Switches inferring the Petri Net in Figure 4.12

On the other hand, if the lack of coordination in f3-2 does not result in deadlock, then two output signals are produced by the system, one through f3-1, resulting in the full use of the two coordinated simple functionalities, and one through f3-2, where the output is produced with reduced processing. In this case, the system does not really accomplish its task, its multiple output signals are probably not consistent.

If lack of coordination results in deadlock, then it is interesting to check that all the simple functionalities do not contain a transition at which deadlock occurs. If it is so, then the system as a whole is deadlocked, and then it cannot accomplish its mission.

4.6 SUMMARY

In this chapter, the different types of shortfalls and overlaps have been defined, and the method to identify them has been presented. Clearly, in the case of a partial or complete shortfall, the system cannot carry its mission in accordance with the requirements. On the other hand, the presence of redundancy with concurrency is beneficial to the operations of the system as it enhances reliability. The case of redundancy with synchronization or of lack of coordination at some function poses some problems; it is difficult to provide a definite answer as to what influence, good or bad, it has on the system's operations. Therefore, the answer to that question lies in the study of the structure of the net on a case by case basis.

CHAPTER V

AN AIR INTERDICTION MISSION SYSTEM

5.1 INTRODUCTION

In this chapter, a system used to plan and execute air interdiction missions is employed to illustrate the methodology evaluating the functionality of systems, against their requirements. The system is called MESACC, which stands for Modular, Endurable, Survivable, Austere, Command Center.

5.2 PROBLEM DEFINITION

The objective of an air interdiction mission system is to **plan** operations against the enemy's military potential before it can be effectively used against friendly forces. These operations restrict the combat capability of the enemy by:

- delaying, disrupting, or destroying their lines of communications
- destroying enemy supplies
- attacking fixed, moving and movable point and area targets
- destroying unengaged or uncommitted enemy attack formations before they can be brought into the battle.

The result of these operations is to disrupt enemy plans and time schedules. The integration of air interdiction operations with the fire and maneuver plans of surface forces is not required. However, these offensive air operations are planned and conducted as part of the unified effort of all friendly forces. Therefore, air interdiction demands precise coordination in timing.

The assumptions and abstractions used in this chapter to model the requirements and the actual implementation pertinent to an air interdiction mission system may not correspond to actual air interdiction operations and do not necessarily reflect real tactical planning and operations.

5.3 FUNCTION IDENTIFICATION

The context consists of the geographical characteristics of the battle area. It is assumed that the system is operating in Europe, and more specifically in the center of the European battlefield (CENTAG).

The environment consists of the friendly forces, their assets, strength, current plans and orders, the enemy forces, their assets, strength, current plans and orders. Also, the current weather is part of the environment as it is a particularly important factor in air interdiction mission planning.

In order to plan an air interdiction mission. the system needs several functions. These functions are presented bellow:

- *weather projection*:
  This function forecasts the weather from the current weather reports sent.
- *format messages*:
  This function transform the format of the various data inputs into a common format. This is necessary as, for security reasons, different coding may exist for various sensors. The function performs also decoding.
- *construct database*:
  This function is needed under certain circumstances, when the system is initialized (beginning of the hostilities)
- *update database*:
  This function updates the current information as new messages comes in the system. For example, if the database contains the position of a particular enemy battalion, and later an intelligence report confirms that the battalion has moved to another position, then the function is used to update the position of that battalion, its strength, and current plans.
- *status of allied forces*:
  This function is used to assess the current state of the allied forces, number of surface troops, equipment, and of available aircraft for missions.
- *strike assessment*:
  This function updates the situation on the battlefield, as a result of previous air interdiction missions.
- *threat assessment*:
  This function evaluates the threat of the enemy forces in the different subareas of the battlefield
- *intelligence report*:
  under certain circumstances, reports from intelligence may be requested when the uncertainty about some parameters of the problem is deemed important.

- *target prioritization/target development*:
  This function is needed to prioritize the most important objective to be destroyed, given the situation. As a matter of fact, the resources that can be used for the next mission may be scarce so that it may not be possible to allocate assets for the destruction of all the objectives.
- *aimpoint construction/weaponeering*:
  This function provides the coordinate of the target, and allocates certain classes of friendly assets according to the objective and its intrinsic characteristics. There are different types of aircraft, each being able to carry different types of weapons
- *penetration/attrition analysis*:
  This function forecasts the degree of redundancy that is adequate for each objective. Different planes may be assigned the same objective, in response to enemy capabilities to destroy the friendly assets in the course of their mission. Therefore, redundancy insures a greater degree of certainty over the outcome of the mission.
- *mission planning*:
  this function delivers the final output of the system to the environment. It consists of a set of missions with the objectives, the type of airplane to be used, its armament, the number of planes to be used for each objective, the route to be followed, and the time to perform the mission.
- *weapon system availability*:
  This function describes what weapons are available for the mission at the time it is planned. This function tells what is available according to the weather forecasts (some aircraft cannot fly under certain circumstances), and the status of the allied forces (losses, use of reserves).

5.4 MESACC: THE REQUIREMENTS

The sensors include satellites, (for the weather reports), the local Command Center which assigns tasks to the system, the regional Tactical Command Center, the Commander IN Chief EURope (CINCEUR), surveillance aircraft planes, and intelligence systems. The various data inputs these sensors deliver are weather reports, reports on friendly and enemy forces (strength, position, status), combat reports, request from the local Command Center for assistance, mission reports, and current and future operation plans.

The single output consists of air interdiction mission plans that are sent to CINCEUR.

The system is considered for a scenario where the hostilities started two days ago. Although the enemy has gained ground on the battlefield, the friendly forces resist the pressure, and major assets in reserve have not been committed on either side. The conflict is a conventional one. The friendly forces and the enemy forces have both fairly accurate information about the situation on the other side. Each side knows what the resources are on the opposing side, as well as the location of these assets, although some uncertainty remains. In certain areas, the battle line is difficult to assess. Therefore, there is a need to use MESACC to plan long distance bombing from high altitude. In this scenario, some functions are not necessary, such as the "construct database" function. Since the conflict started two days ago, the database already exists. All the other functions described above are in use. Their labeling is as follows:

f1: Weather Projections
f2: Format Messages/Fusion of Information/Update Database
f3: Status of Allied Forces
f4: Strike Assessment
f5: Threat Assessment
f6: Current Intelligence
f7: Target Prioritization/Target Development
f8: Aimpoint Construction/Weaponeering
f9: Penetration/Attrition Analysis
f10: Mission Planning
f11: Weapon System Availability

The interrelationship between the various functions is: the mission planning function can be performed after the Aimpoint Construction/Weaponeering function, the Penetration/Attrition Analysis function, the Weapon System Availability function, and the Weather Projection function. the Aimpoint Construction/Weaponeering function is derived from the Target Prioritization/Target Development function, which is itself derived from the Strike Assessment function. The Penetration/Attrition Analysis function is also derived from the Strike Assessment function. The Strike Assessment function is derived from the Threat Assessment function, and eventually the Current Intelligence function. Lastly, the Weapon System Availability function is derived from the Weather Projection function, and the Status of Allied Forces function. It should be understood that this description of the interrelationship between functions is purely functional. If a function is derived from another, it does not mean that the input of that function is sufficient. Indeed, data from the context may be necessary (terrain information for example).

The description of the functional requirements of MESACC is depicted by the Petri Net of Figure 5.1.

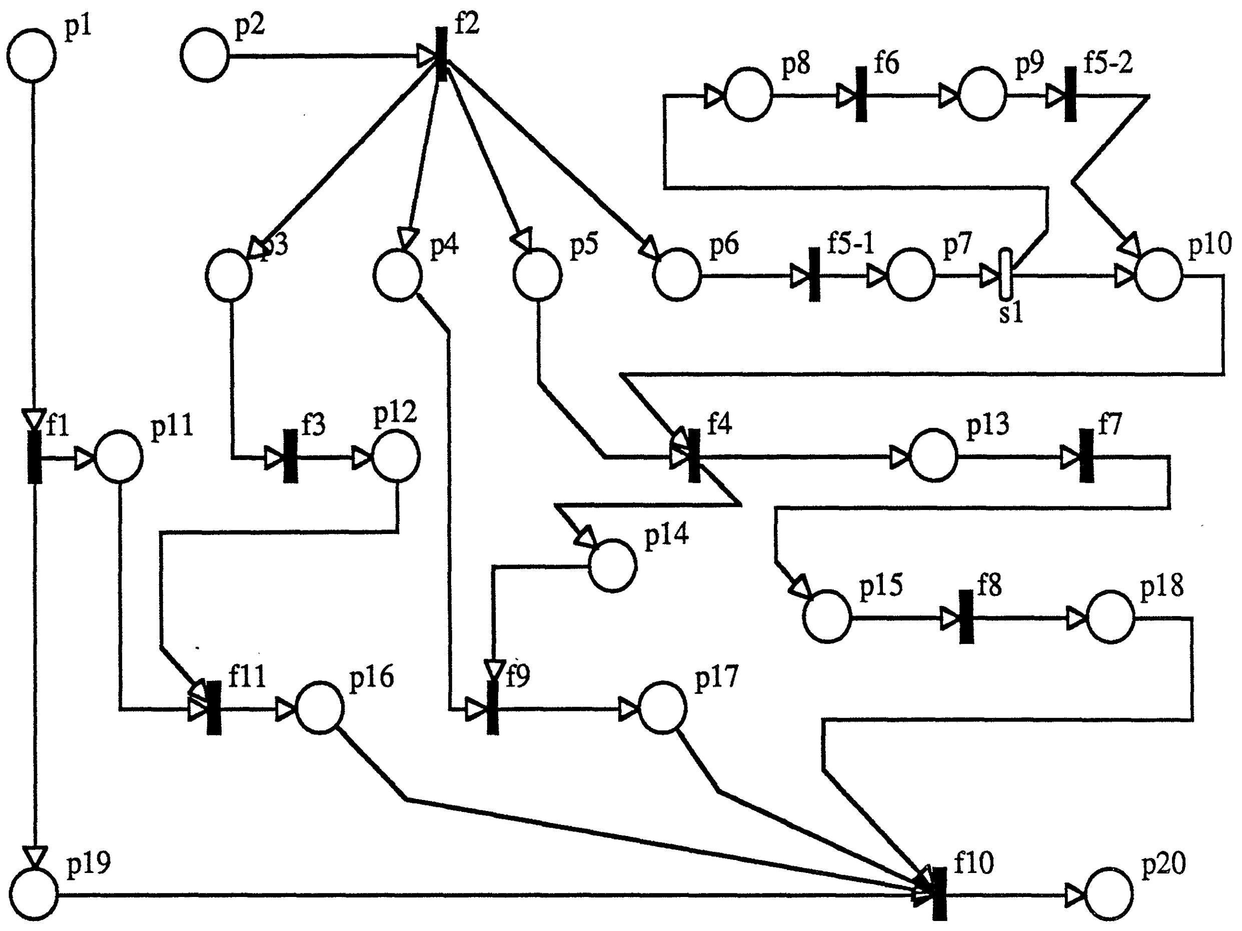


Figure 5.1 MESACC: Functional Requirements

This Requirements Net contain one switch only. This switch has two output branches. The switch represents the use or the non use of the "Current Intelligence" function. Therefore, from that Petri Net, two ordinary Petri Nets are derived by removing the switch, according to the algorithm described in Chapter III, each one corresponding to a different strategy.

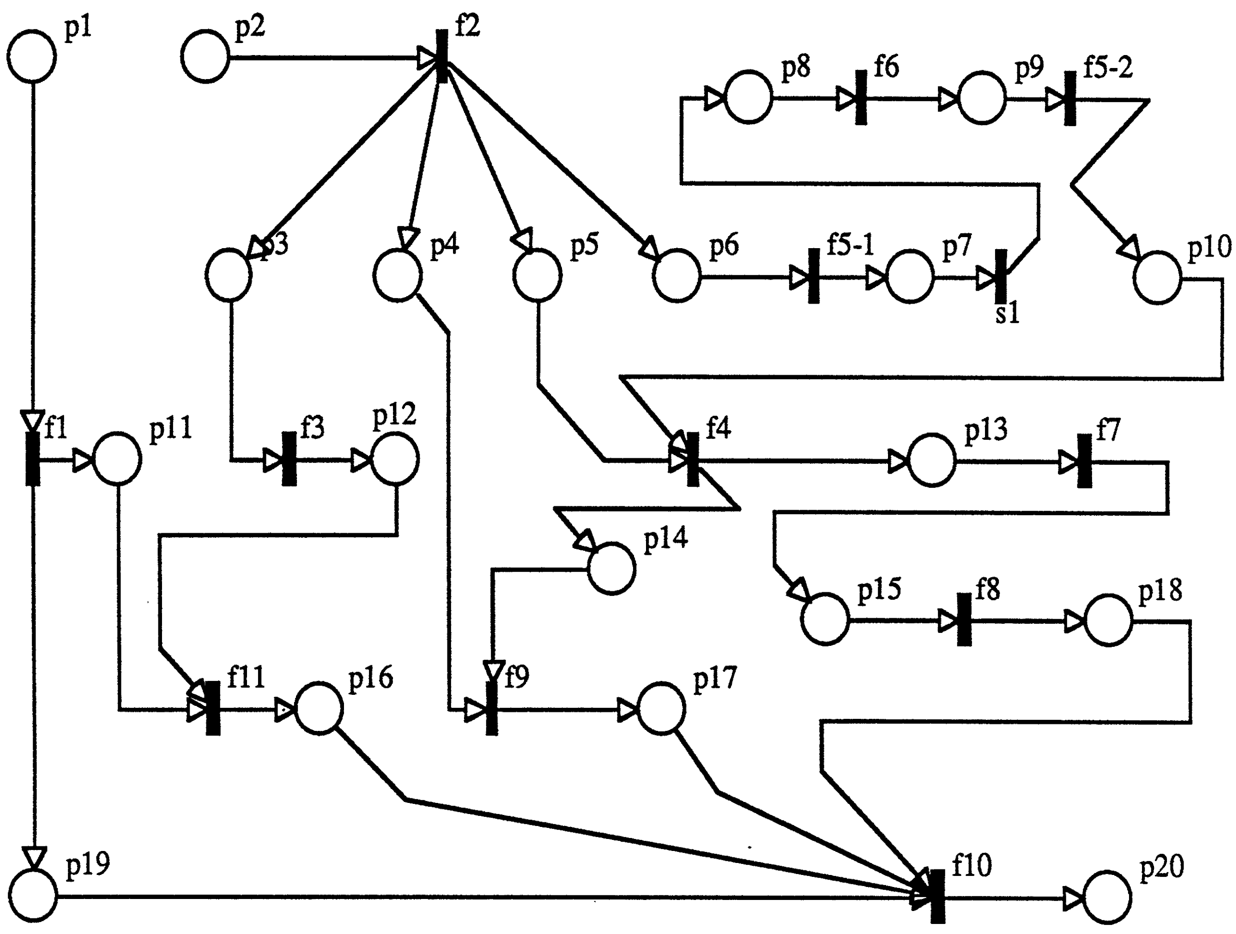


Figure 5.2 MESACC: Strategy One, f6 is used

5.5 MESACC: THE IMPLEMENTED SYSTEM

The implemented system is composed of a number of physical components. In the following description, the scenario is already taken into account. The scenario is of importance in the description of some of the part of the systems, such as the number and nature of the functions embedded in the workstations, and their interrelationship.

5.5.1 Physical Components

- It has been assumed that the physical system is geographically distributed. There are four shelters:

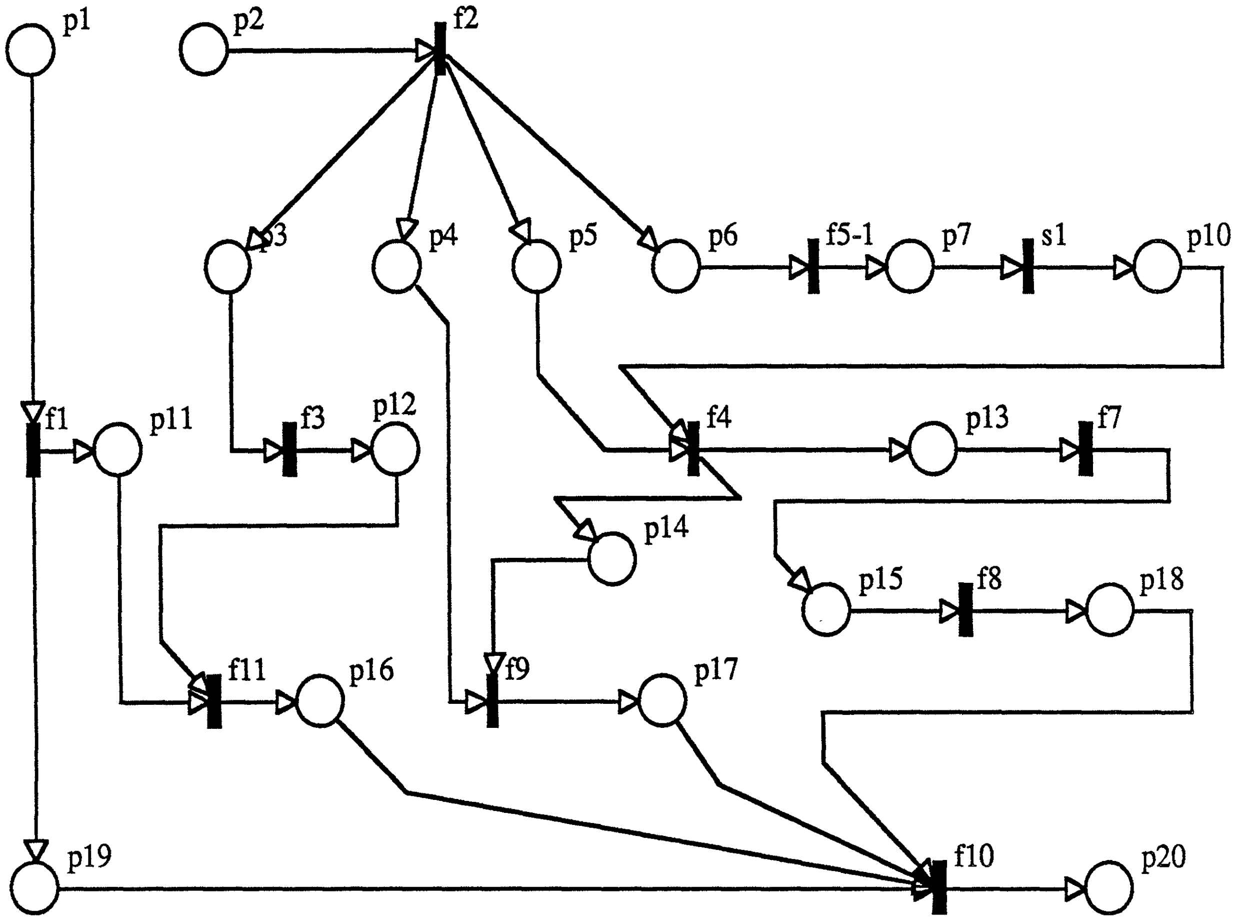


Figure 5.3 MESACC: Strategy Two, no use of f6

- the Battle Management shelter (BM shelter). This shelter contains the functions that are directly connected to the battle itself, such as "Target Prioritization/Target Development".

- the Intelligence / War Planning shelter (I/WP) shelter. This shelter contains functions that are more concerned with the status of the war in general, such as "Status of Allied Forces" as well as functions using information from intelligence sources.

- the Intelligence / War Planning / Automated Data Processing shelter (ADP shelter). This shelter contains all the equipment necessary for the automated treatment of data. In particular, this shelter contains the database.

- the INput Communication shelter (INC shelter). This shelter receives all the inputs from the different sensors of the system, using different communication means. It is possible for a sensor to send the same item of information by different communication means. These means are:
  - radio HF/VHF
  - telephone line
  - satellite (as a communication relay)

- The functions are also distributed into eleven workstations:
  These workstations are embedded in the BM shelter and the I/WP shelter. There are five workstations in the I/WP shelter and six workstations in the BM shelter. Table 5.1 describes which workstation carries which functions. The labeling of the functions is the one adopted in section 5.2. Workstations A1 to A6 are located in the BM shelter, while workstations A7 to A11 are located in the I/WP shelter.

Table 5.1 Assignment of Functions to the Workstations of MESACC

	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11
f1	1				1						
f2							1	1			1
f3							1				
f4								1	1		
f5									1		
f6											1
f7		1				1					
f8		1		1		1					
f9			1			1				1	
f10			1	1						1	
f11					1						

Each workstation includes, typically, a CPU with its associated software, and the capability to perform the functions assigned to it. Most of the workstations may perform different functions, while one worksation is able to perform one function only (A1 can perform f1 only).

5.5.2 Boundaries of the System

To determine the boundaries of the system, it is sufficient to describe the context and the environment.

The context is made of:

- the enemy forces
- the weather
- the terrain, i.e., the geography of the battle area.

The environment is made of:

- The allied forces
- the intelligence sources, and more precisely, the JTIDS (Joint Tactical Information Distribution System).
- CINCEUR
- the satellites (as message relays and sensor devices).

5.5.3 Message Inputs

Among the wide varieties of messages that such a system can receive, fourteen critical messages have been identified. These fourteen messages are described in detail as follows:

- c1: Weather reports
- c2: Enemy reserve forces report:
    - date of report
    - location
    - strength
    - functional capabilities
    - movement status and capabilities
- c3: Enemy engaged forces report:
    - date of report
    - location
    - strength
    - functional capabilities
- c4: Enemy forces, not engaged but close to the FEBA (Forward Edge of the Battle Area) report:
    - date of report

    - location
    - strength
    - functional capabilities
    - estimated intention
- c5: Enemy support status report:
    - date of report
    - location
    - nature of support (ammunitions, fuel, water, food, spare parts, hardware)
- c6: Allied reserved forces:
    - date of report
    - location
    - strength
    - functional capabilities
    - movement status and capabilities
- c7: Allied engaged forces report:
    - date of report
    - location
    - strength
    - functional capabilities
- c8: Allied forces, not engaged but close to the FEBA report:
    - date of report
    - location
    - strength
    - functional capabilities
    - mission orders and plan
- c9: Allied support status report:
    - date of report
    - location
    - nature of support (ammunitions, fuel, water, food, spare parts, hardware)
    - mission
- c10: Combat report:
    - date of report
    - unit involved
    - location

  - losses
  - enemy losses (estimates)
  - local situation update (geographic repositioning, state of the unit)
- c11: Immediate combat report:
  - date of report
  - status
  - request for support (ammunition, food, water)
- c12: Mission report:
  - date of report
  - mission I.D.
  - result, and confidence factor on the result
  - Allied losses
- c13: Current regional operations orders and plans:
  - date of report
  - objectives
  - priority of objectives
  - units involved
- c14: Future regional operations orders and plans:
  - date of report
  - objectives
  - priority of objectives
  - units involved

5.5.4 Interrelationship between Functions

Table 5.2 shows which messages are needed by which function. The messages are in the row of the matrix, while the functions are in the column of the matrix. A "1" in a cell of the matrix located at the ith row and at thee jth column of the matrix indicates that the message c_i is critical for f_j to act in a proper fashion.

Table 5.3 shows the relation between the different functions. The rows and the columns of the matrix represent the functions. A "1" in a cell of the matrix at the ith row and jth column indicates that when function f_i is completed, then function f_j can start its process, because f_j can obtain the result of f_i.

The same function may be processed at different workstations. This is done to increase the reliability of the system. Also, that same function may be implemented on different workstations, which in turn are located in different shelters. This is done to enhance the survivability of the system, so that if one of the shelters becomes inoperable, the system can still carry out its mission. As a result of this process redundancy, other tables are needed to show in greater detail the interrelationship between the different processes.

Table 5.2 Criticality of Messages by Functions

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11
c1	1										
c2		1		1	1	1	1	1		1	
c3		1		1	1		1	1		1	
c4		1		1	1	1	1	1		1	
c5		1		1	1	1	1	1		1	
c6		1	1						1	1	1
c7		1	1						1	1	1
c8		1	1						1	1	1
c9		1	1						1	1	1
c10		1	1		1		1			1	1
c11		1	1		1		1			1	
c12		1	1						1	1	1
c13		1					1		1	1	1
c14		1					1		1	1	1

Table 5.3 Relations between Functions

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11
f1										1	1
f2		1	1	1	1	1			1		
f3											1
f4							1		1	1	
f5				1		1			1	1	
f6		1		1	1					1	
f7								1		1	
f8										1	
f9										1	
f10										1	
f11										1	

Table 5.4 shows the interrelationship between the functions in the system, and the functions embedded in the BM shelter. All the functions are represented in rows, with their workstation location. Each column corresponds to a function in the BM shelter with its workstation location. A "1" in the ith row and jth column of the matrix indicates that when function f_i is completed, the result of its processing is available for the function f_j. Table 5.5 shows the same information for the functions located in the I/WP shelter.

Table 5.4 Relations Between Functions and Functions in the BM Shelter

		BM Shelter											
		A1	A2		A3		A4		A5		A6		
		f1	f7	f8	f9	f10	f8	f10	f1	f11	f7	f8	f9
A1	f1					1		1		1			
A2	f7			1		1	1	1				1	
A2	f8					1		1					
A3	f9					1		1					
A4	f8					1		1					
A4	f10					1		1					
A5	f1					1		1					
A5	f11					1		1					
A6	f7			1		1	1	1				1	
A6	f8					1		1					
A6	f9					1		1					
A7	f2					1		1		1			
A7	f3					1		1		1			
A8	f2					1		1		1			
A8	f4		1		1						1		1
A9	f4		1		1						1		1
A9	f5												
A10	f9					1		1					
A10	f10					1		1					
A11	f2					1		1		1			
A11	f6												

This relation between functions can be effected directly, through the transmission of a message from a function to the other. It can also be done through the use of a database: the function sends the result of its processing to a database which stores the information. The other function accesses that information through a request to the same database. There are 7 databases in the implementation of MESACC, labeled DB1 to DB7. The description of the content of each one is presented below:

- DB1: contains weather forecasts that come from the weather reports and which are produced by f1.
- DB2: contains data about the enemy i.e., the contents of c2 to c5.
- DB3: contains data about the allied forces i.e., the contents of c6 to c9.
- DB4: contains data about the situation on the battle field, i.e., the contents of c10 to c13.
- DB5: contains data about threat assessment, i.e., the results of f5.
- DB6: contains data about weapons availability, given the status of the allied equipment, and the weather forecasts.
- DB7: contains data about strike assessment, i.e., the result of f4.

Table 5.5 Relations Between Functions and Functions in the I/WP Shelter

		I/WP Shelter									
		A7		A8		A9		A10		A11	
		f2	f3	f2	f4	f4	f5	f9	f10	f2	f6
A1	f1								1		
A2	f7										
A2	f8										
A3	f9										
A4	f8										
A4	f10										
A5	f1										
A5	f11										
A6	f7										
A6	f8										
A6	f9										
A7	f2		1		1	1	1	1			1
A7	f3										
A8	f2		1		1	1	1	1			1
A8	f9								1		
A9	f2		1		1	1	1	1			1
A9	f4							1			
A10	f4							1			
A10	f5				1	1	1				1
A11	f9								1		
A11	f10								1		
A12	f2		1		1	1	1	1			1
A12	f6	1		1			1			1	

These databases are accessed by different functions. Table 5.6 shows which function requests data from which database. Each database is represented by a row, while each function is represented by a column, with its number of occurrence in the system. A "1" in the ith row and jth column of the matrix indicates that f_j requests data from DB_i.

5.5.5 The Implemented System

The whole system is represented in Figure 5.4. The shelters are represented by the polygons in light shaded gray, while the workstations are represented in dark shaded gray. The labeling of the places and transitions of the net is as follows:

- The places labeled c1 to c14 correspond to the input messages c1 to c14.
- The transition labeled fi-j-k corresponds to the kth occurrence of an fi function. This function may be decomposed in various subfunctions, and j denotes the jth subfunction of that particular fi function.
- The place labeled ji corresponds to the output of an fi function that is sent to another function.
- The transition labeled E-DBi corresponds to the entrance to the ith database.
- The transition labeled up-i-j corresponds to the update of the jth occurrence of the ith database.
- The transition labeled gd-i-j corresponds to the reception of the query and the passing of data of the jth occurrence of the ith database. This happens when a certain function requests the data from that database.
- The transition labeled ir-i-j corresponds to the availability of the data of the jth occurrence of the ith database. This occurs when a query for data has just been made.

Table 5.6 Request to Database from the Functions in MESACC

Function	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	total
# in system	2	3	1	2	1	1	2	3	3	3	1	22
DBASE #1										1	1	4
DBASE #2			1	1	1			1				7
DBASE #3							1	1				5
DBASE #4							1					2
DBASE #5				1								2
DBASE #6										1		3
DBASE #7									1			3

This System Net contains five switches. Each of the five switches has two output transitions. However, switches s7 and s8 are cascaded with switch s6. Therefore, the number of different switch settings is 20. There are twenty different System Strategy Nets, each corresponding to a different switch setting. Table 5.7 provides a labeling of the different Petri Nets, in correspondence with its switch setting. The five switches are represented in the first five columns. An ordinary Petri Net representing a strategy is given by the unique output place of each switch considered in a given strategy. In some occurrences, a dash indicates that this switch setting is irrelevant because it is cascaded with another switch; this is because that switch is included in an output branch of another switch which setting does not consider that branch.

Table 5.7 The twenty Different Strategies of MESACC

Switches					Strategy #
s2	f5-3-1	s6	s7	s8	
p7	p191	p124	-	-	1
p7	p191	p125	p195	p194	2
p7	p191	p125	p195	p123	3
p7	p191	p125	p127	p194	4
p7	p191	p125	p127	p123	5
p8	p191	p124	-	-	6
p8	p191	p125	p195	p194	7
p8	p191	p125	p195	p123	8
p8	p191	p125	p127	p194	9
p8	p191	p125	p127	p123	10
p7	p192	p124	-	-	11
p7	p192	p125	p195	p194	12
p7	p192	p125	p195	p123	13
p7	p192	p125	p127	p194	14
p7	p192	p125	p127	p123	15
p8	p192	p124	-	-	16
p8	p192	p125	p195	p194	17
p8	p192	p125	p195	p123	18
p8	p192	p125	p127	p194	19
p8	p192	p125	p127	p123	20

5.6 EVALUATION OF FUNCTIONALITY

5.6.1 Methodology

Given that there are two Required Strategy Nets in the requirements and twenty System Strategy Nets in the implementation of MESACC, the first thing to do is to relate each of the twenty to one of the two Required Strategy Nets. The difference between the two Required

Strategy Nets lies in the use of the function "Current Intelligence", labeled f6. The Strategy Nets of the implemented system labeled 1 to 10 do not use f6, as shown by the f5-3-1 switch setting which branches with p191. Therefore each of these nets is to be compared with Strategy 2 Required Petri Net. On the other hand, the System Strategy Nets labeled 11 to 20 use f6, as shown by the f5-3-1 switch setting which branches with p192. Therefore each of these Petri Nets is to be compared with Strategy 1 Required Petri Net.

A functional evaluation will be performed now in detail between Strategy 2 Required Petri Net and Strategy 1 System Petri Net. Then the results of the evaluation of the system against its requirements will be presented.

5.6.2 Complete Functionality in the Required Strategy Net

The computation of minimal support S-Invariants in the Required Strategy Net reveals the existence of eight minimal support S-Invariants. The support of these invariants is presented below:

- Invariant 1: p1 p19 p20
- Invariant 2: p2 p5 p13 p15 p18 p20
- Invariant 3: p2 p4 p17 p20
- Invariant 4: p2 p5 p14 p17 p20
- Invariant 5: p1 p11 p16 p20
- Invariant 6: p2 p3 p12 p16 p20
- Invariant 7: p2 p6 p7 p8 p9 p10 p13 p15 p18 p20
- Invariant 8: p2 p6 p7 p8 p9 p10 p14 p17 p20

As there is only one sink in the net, there is only one complete functionality. Each of the minimal support S-Invariants contains the sink, p20, in its support. Therefore, the complete functionality of the required system is made of the coalescence of the S-Components of all the minimal support S-Invariants. Since the net is covered by the S-Invariants, i.e., any place of the net belongs to at least one minimal support S-Invariant, the complete functionality is represented by the whole net, as depicted in Figure 5.3.

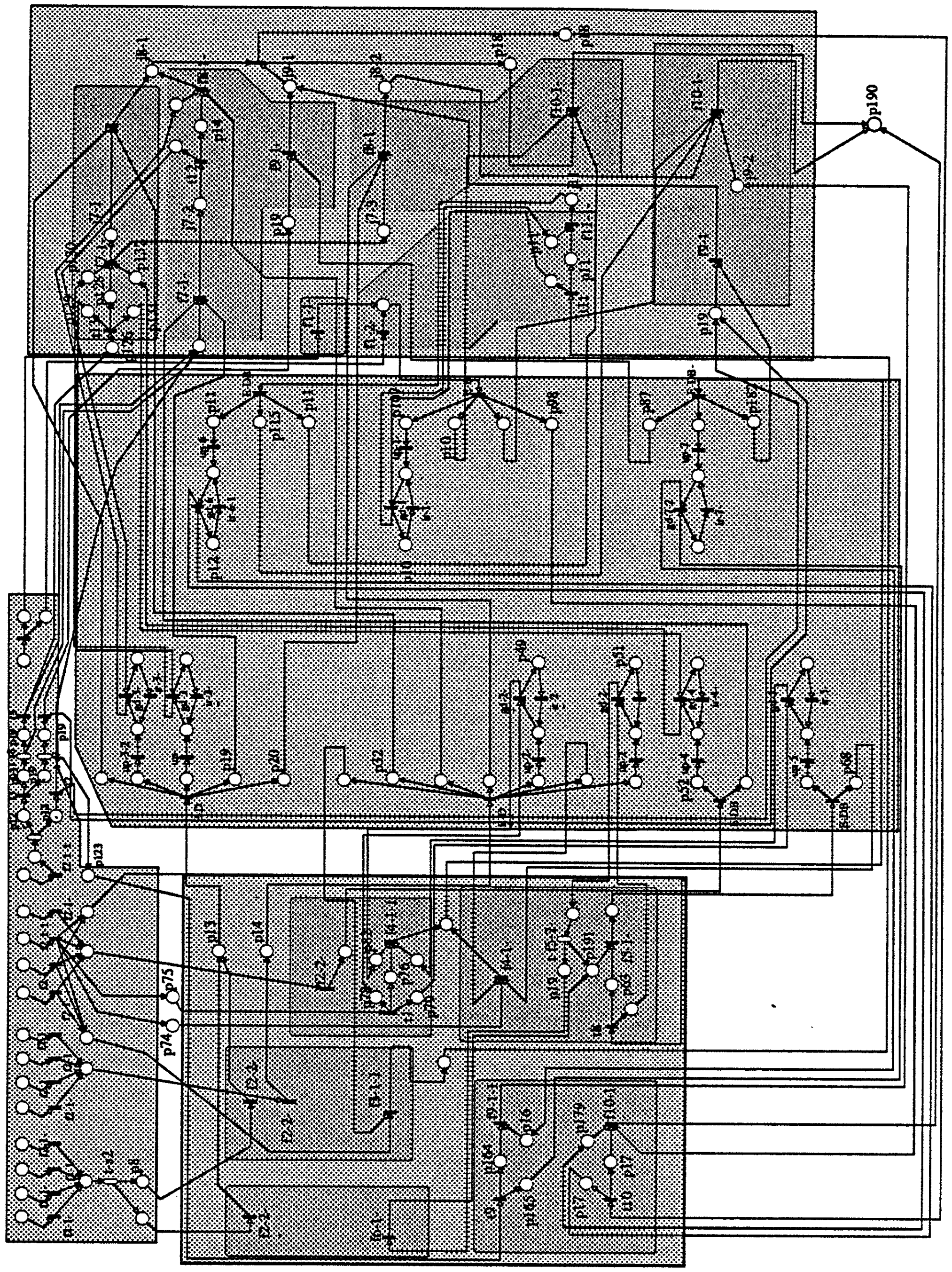


Figure 5.4 MESACC: The whole System Net

5.6.3 Simple Functionality in the Requirements Petri Net

After the complete functionality has been identified, the next step is to construct the simple functionalities within the complete functionality. Since the complete functionality is a marked graph, each simple functionality corresponds to a minimal support S-Invariant. Therefore, there are eight simple functionalities that are listed below:

- Pa1: p1 | f1 → f10
- Pa2: p1 | f1 → f11 → f10
- Pa3: p2 | f2 → f3 → f11 → f10
- Pa4: p2 | f2 → f9 → f10
- Pa5: p2 | f2 → f4 → f9 → f10
- Pa6: p2 | f2 → f4 → f7 → f8 → f10
- Pa7: p2 | f2 → f5-1 → f4 → f9 → f10
- Pa8: p2 | f2 → f5-1 → f4 → f7 → f8 → f10

5.6.4 Complete Functionality in the Strategy System Net

The computation of minimal support S-Invariants in the System Strategy Net reveals the existence of 275 minimal support S-Invariants. The supports of these invariants are presented in Appendix C. Since there is only one sink in the net, there is only one complete functionality. The construction of the complete functionality is done in the two steps described in chapter III.

In the first step, the S-Components of the minimal support S-Invariants that contain the sink, p190, are coalesced together to form a subnet. There are 265 out of the 275 minimal support S-Invariants which contain the sink in their support. Therefore the S-Components of these 265 minimal support S-Invariants are to be taken into account in the construction of the complete functionality. The subnet obtained is depicted in Figure 5.5.

In the second step of the construction, the transitions of these S-Components are scanned to check that they have the same number of input places in that subnet as in the System Strategy Net. If this is not the case, that is, there is an input place of one of these transitions which is not present in the constructed subnet, then any minimal support S-Invariant that contains that input place in its support is coalesced in the subnet. Again, the same rule of addition of minimal support S-Invariant is applied to the transitions of the previously added S-Components, and so forth until it is no longer possible to add any S-Component of a minimal support S-Invariant. For the net obtained from the first step, the transitions fulfilling the condition are:

gd-1-1, gd-2-5, gd-2-7, gd-3-2, gd-5-1, gd-6-1.

Note that every one of these transitions belongs to an information loop, although it is not a requirement for applying the construction method. In the following, all the S-Components of minimal support S-Invariants which contain these transitions are added to the subnet. They correspond to the following minimal support S-Invariants:

for gd-1-1: the minimal support S-Invariant labeled 129
for gd-2-5: the minimal support S-Invariant labeled 1
for gd-2-7: the minimal support S-Invariant labeled 2
for gd-3-2: the minimal support S-Invariant labeled 170
for gd-5-1: the minimal support S-Invariants labeled 3, and 197
for gd-6-1: the minimal support S-Invariants labeled 4, 5, 6, and 7

Therefore, the S-Components corresponding to these minimal support S-Invariants are added. For these sub-nets, all the transitions have the same number of input places than in the net. Further this sub-net is covered by the S-Invariants, i.e. any place of the net belongs to at least one minimal support S-Invariant; there is no need to check for other nodes that would belong to the complete functionality. The adding method of minimal support S-Invariant is sufficient. The complete functionality is now obtained and is depicted in Figure 5.6.

5.6.5 Simple Functionality in the System Strategy Net

The algorithm presented in chapter III is applied here to determine all the simple information flow paths. There are 479 simple information flow paths. By applying the discrimination method described in chapter V, the number of relevant simple information flow paths is reduced to 102.

These simple information flow paths are presented in appendix D.

Complete shortfall: There is a single sink in the Required and in the System Petri Net. Since they correspond, there is no complete shortfall.

Partial shortfall: In order to identify partial shortfalls, the equivalence between sources and functions has to be made. Table 5.8 presents the equivalence between sources in the two Petri Nets, while tables 5.9 presents the equivalence between functions across the two Petri Nets.

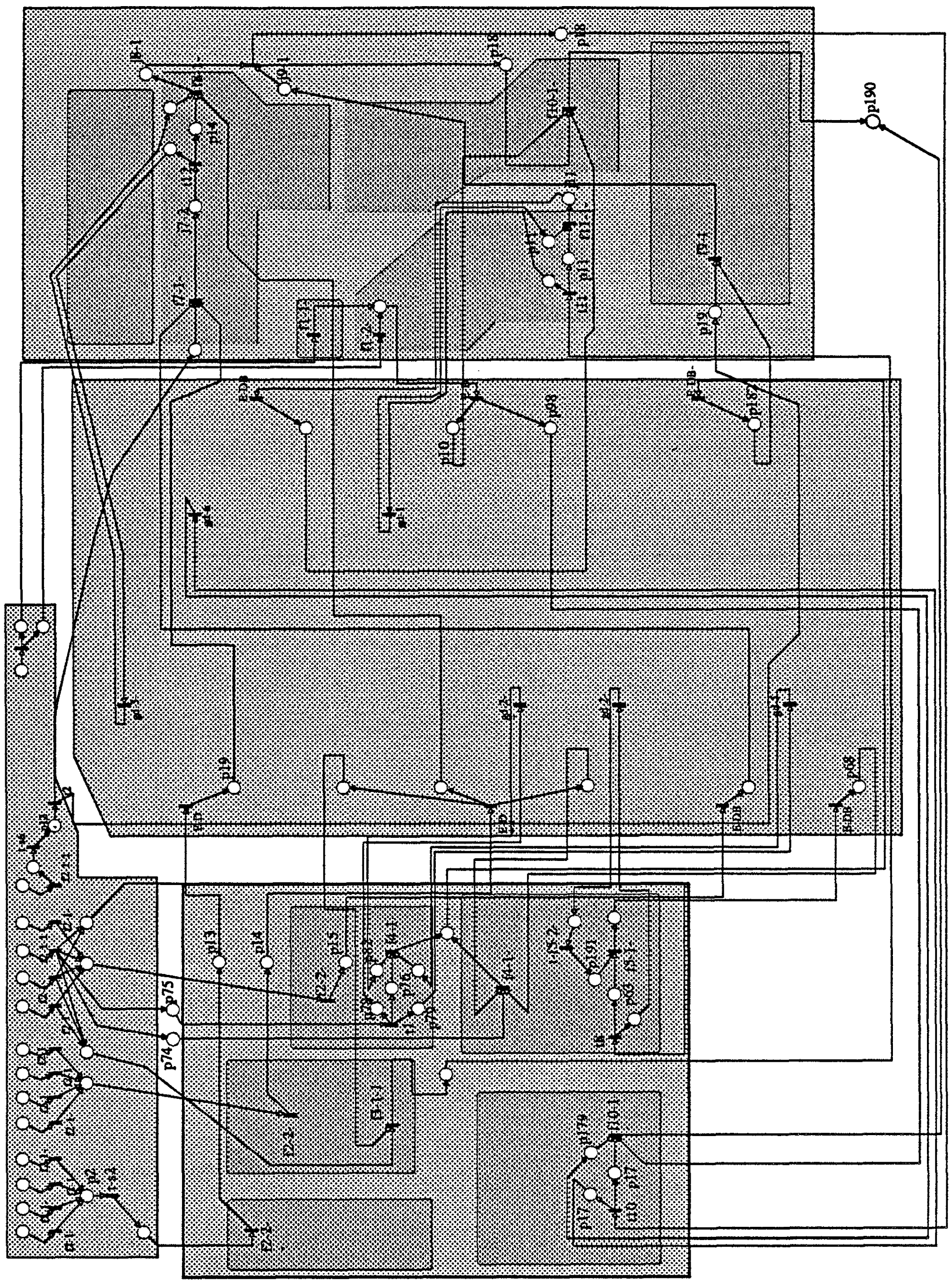


Figure 5.5 MESACC: First step of the Complete Functionality construction for System Net #1

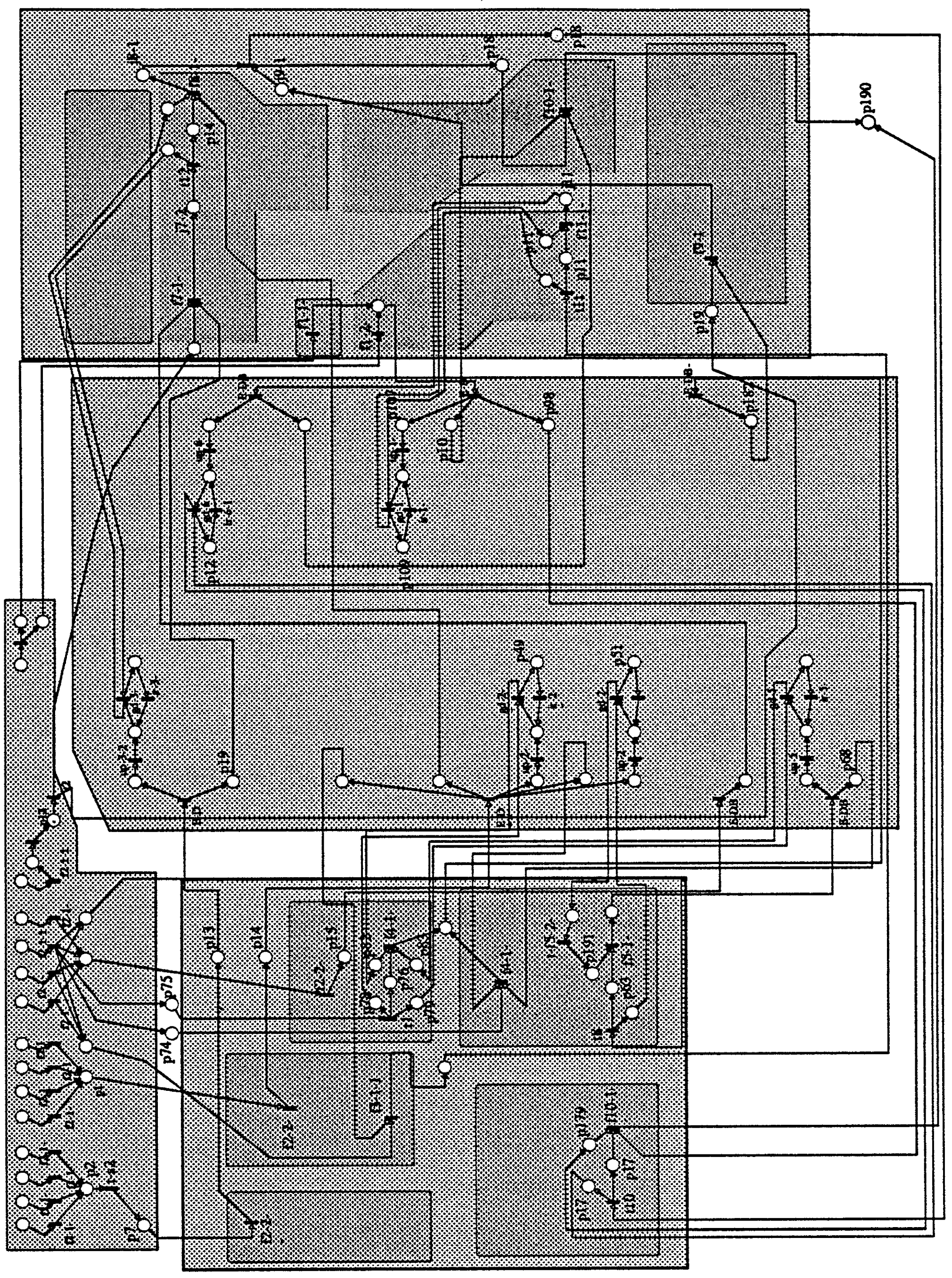


Figure 5.6 MESACC: Complete Functionality for System Net #1

Table 5.8 Equivalence between Sources

Requirements Sources	System Sources
p1	c1
p2	c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14

It is now possible to proceed with the analysis of partial shortfalls. For each simple functionality in the requirements net that corresponds to a simple information flow path, a corresponding simple information flow path is searched in the System Strategy Net, i.e., a simple information flow path having the same sequence of equivalent functions. Table 5.10 provides the result of this search.

As the table shows, there is a partial shortfall for the simple information flow paths Pa6 and Pa8. On the other hand, there are eighteen simple information flow paths in the System Net which have a sequence of functions equivalent to: f2 → f7 → f8 → f10, and sixteen simple information flow paths which have a sequence of functions equivalent to: f2 → f8 → f10. These simple functionalities cannot be related to the required simple functionalities.

For the remaining six simple functionalities that have equivalent counterparts in the System Strategy net, the next step is to compare the inputs they use. It is now possible to construct another table in which the input sources of each simple information flow path can be compared with the sources of its required equivalent. Given the correspondence between sources established in table 5.8, it is possible to determine if the same inputs are used by the equivalent simple functionalities. Table 5.11 shows that relation. In its first column, the input source of the required simple functionality is shown. In the following columns, the number of equivalent simple functionalities using the input source on top of the column is shown. The last column determines if the required functionality uses input sources contained in the input sources used by the equivalent implemented simple functionalities.

Therefore, only Pa1 and Pa2 have strictly equivalent counterparts in the implemented system. Pa3, to Pa6 have equivalent functional paths, but these paths do not use all the sources. The question is to determine if the non use of certain sources in the implemented system simple functionalities constitute a shortfall or not.

Table 5.9 Equivalence between Functions

Requirements Functions	System Functions
f1	f1-1-1 f1-2-1
f2	f2-1-1 -> f2-2-3 f2-1-2 -> f2-2-3 f2-1-3 -> f2-2-3 f2-1-4 -> f2-2-3 f2-1-5 -> f2-2-1 f2-1-6 -> f2-2-1 f2-1-7 -> f2-2-1 f2-1-8 -> f2-2-1 f2-1-5 -> f2-2-2 f2-1-6 -> f2-2-2 f2-1-7 -> f2-2-2 f2-1-8 -> f2-2-2 f2-1-9 ->f2-2-4 f2-1-10 ->f2-2-4 f2-1-11 ->f2-2-4 f2-1-12 ->f2-2-4 f2-1-13
f3	f3-1-1
f4	f4-1-1 f4-1-2
f5-1	f5-2-1
f7	f7-1-1 f7-1-2
f8	f8-1-1 f8-1-2 f8-1-3
f9	f9-1-1 f9-1-2 f9-1-3
f10	f10-1-1 f10-1-2 f10-1-3
f11	f11-1-1

For Pa3, the use of messages c6 to c9 seems critical as the first function in that simple functionality is f3, i.e. "status of allied forces". Therefore there is a problem of inputs.

Table 5.10 Partial Shortfalls

Requirements Simple information flow path	Number of equivalent system simple information flow paths
Pa1	4
Pa2	4
Pa3	14
Pa4	2
Pa5	20
Pa6	0
Pa7	24
Pa8	0

Table 5.11 Equivalence of input usage

Simple functionality	Input source	Input source in the implemented system net														Equivalence
		c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	
Pa1	p1	4														YES
Pa2	p1	4														YES
Pa3	p2		2	2	2	2					2	2	2			NO
Pa4	p2														2	NO
Pa5	p2		4	4	4	4							4			NO
Pa7	p2		4	4	4	4						4	4			NO

For Pa4, the use of c6 to c9 may not be critical; on the other hand the use of c10 and c11 is critical, as the first function to be performed is f9, i.e., Penetration/Attrition Analysis. Therefore, there is an input problem.

For Pa5, the use of c6 to c9 is not critical; on the other hand the use of c10 and c11, i.e., "combat reports" and "immediate combat report" are critical to perform the first function in the sequence, i.e., "Strike Assessment". Therefore there is a problem of inputs.

For Pa7, the comments are the same as for Pa5.

Therefore, there are two partial shortfalls, four simple functionalities that do not use all the input sources that they should, and two perfectly equivalent simple functionalities.

Redundancies

Table 5.10 and 5.11 provide a basis for finding eventual redundancies in the System Strategy Net. Table 5.10 gives the number of simple information flow paths that have the same sequence of equivalent functions. Table 5.12 provides the number of simple information flow paths that are equivalent, i.e., same source and same sequence of functions. The numbers in the column c1 to c14 are the number of groups of equivalent simple information flow paths. There are twenty one groups of equivalent simple functionalities. For reasons of simplicity, each group of equivalent simple functionalities is labeled by using the name of its common source, and the name of its equivalent simple functionality in the requirements net. A table is constructed that shows the type of redundancy for the simple functionality of each group.

Table 5.12 Redundancies in the implemented Petri Net #1

Group of equivalent simple functionalities		Number of simple functionalities	Nature of redundancy
Equivalent requirements simple functionality	input source		
Pa1	c1	4	concurrency
Pa2	c1	4	concurrency
Pa3	c2	2	concurrency
Pa3	c3	2	concurrency
Pa3	c4	2	concurrency
Pa3	c5	2	concurrency
Pa3	c10	2	concurrency
Pa3	c11	2	concurrency
Pa3	c12	2	concurrency
Pa4	c14	2	concurrency
Pa5	c2	4	double concurrency
Pa5	c3	4	double concurrency
Pa5	c4	4	double concurrency
Pa5	c5	4	double concurrency
Pa5	c12	4	double concurrency
Pa7	c2	4	double concurrency
Pa7	c3	4	double concurrency
Pa7	c4	4	double concurrency
Pa7	c5	4	double concurrency
Pa7	c11	4	double concurrency
Pa7	c12	4	double concurrency

Note that a "double concurrency" means that there are two sequences of concurrent redundancies, as in the Petri Net shown in Figure 5.7.

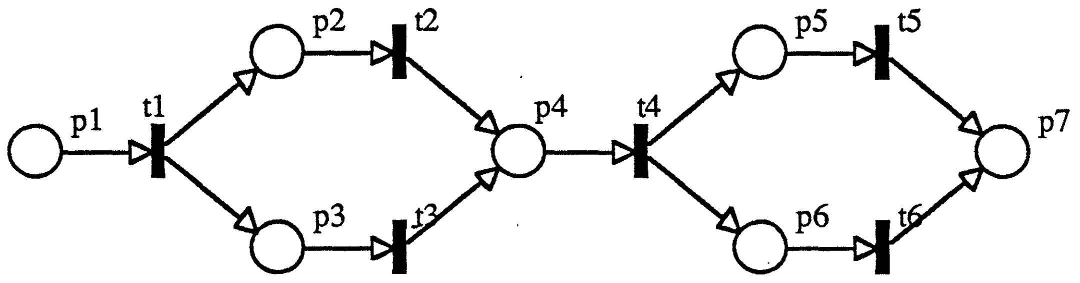


Figure 5.7 An example of Double Redundancy

Therefore, in Strategy #1 System Net, there are redundancies with concurrency only.

Coordination

Table 5.13 shows, for each function, what the functionalities are that contain it. The same table is constructed for the Strategy #1 System Net, table 5.14, in which the representation of the system simple functionalities uses the same notation as in table 5.12. Note that the subfunctions of f2 are not included. This is deliberate, since in the system net every simple functionality uses f2, except for the weather reports, i.e., message c1.

Table 5.13 Functions contained in the required Functionality

Function	Simple functionality							
	Pa1	Pa2	Pa3	Pa4	Pa5	Pa6	Pa7	Pa8
f1	√	√						
f2			√	√	√	√	√	√
f3			√					
f4					√	√	√	√
f5							√	√
f7						√		√
f8						√		√
f9				√	√		√	
f10	√	√	√	√	√	√	√	√
f11		√	√					

Table 5.14 System Functions contained in the System's Simple Functionalities

Functionality		f1-1-1	f1-1-2	f3-1-1	f4-1-1	f4-1-2	f5-1-1	f7-1-2	f8-1-2	f9-1-3	f10-1-1	f10-1-2	f11-1-1
Pa1	c1	√	√								√	√	
Pa2	c1	√	√								√	√	√
Pa3	c2			√							√	√	√
Pa3	c3			√							√	√	√
Pa3	c4			√							√	√	√
Pa3	c5			√							√	√	√
Pa3	c10			√							√	√	√
Pa3	c11			√							√	√	√
Pa3	c12			√							√	√	√
Pa4	c14									√	√	√	
Pa5	c2				√	√				√	√	√	
Pa5	c3				√	√				√	√	√	
Pa5	c4				√	√				√	√	√	
Pa5	c5				√	√				√	√	√	
Pa5	c12				√	√				√	√	√	
Pa7	c2				√	√	√			√	√	√	
Pa7	c3				√	√	√			√	√	√	
Pa7	c4				√	√	√			√	√	√	
Pa7	c5				√	√	√			√	√	√	
Pa7	c11				√	√	√			√	√	√	
Pa7	c12				√	√	√			√	√	√	

Given the two tables, and the equivalence relationship between functions across the net, it is now possible to evaluate the lack of coordination. First of all, by inspecting the incidence matrix, no deadlock appears in this net since every transition has the same number of input places in the Strategy 1 System Petri Net as in the System Net. Second, it appears that all the simple functionalities that use a given function in the Required Strategy Net have an equivalent simple functionality in the System Strategy Net that uses the same equivalent function. Therefore, there is no problem of coordination in the System Strategy Net, except of course due to the existence of partial shortfalls.

Summary

This evaluation of functionality reveals:

- the existence of two partial shortfalls, for Pa6 and Pa8.
- the lack of use of some critical inputs for Pa3, Pa4, Pa5, and Pa7
- the existence of two simple functionalities, namely Pa1, and Pa2, having equivalent

counterparts in the system net.

However, there are two sets of sequences of functions in the implemented system which do not have any counterpart in the requirements. As these sequences of function are close to the sequence of functions of the partial shortfalls, it may not be difficult to derive from the existing structures a structure that would be closer to the requirements.

Also, there are no problems of redundancy, there is no redundancy with conflict, and all the existing redundancy is with concurrency. Thus the reliability and the survivability of that structure is high.

Lastly, there are no deadlocks in the system, and no problems of coordination between the simple functionalities other than those created by the partial shortfalls.

Now that a detailed evaluation has been provided for one pair of Petri Net, the result of the global evaluation of the implemented MESACC against its requirements is presented in the next section.

5.6.6 Evaluation of MESACC

In this section the result of the evaluation of the twenty System Strategy Nets of MESACC against the two Required Strategy Nets is presented.

COMPLETE SHORTFALL

Since each strategy Petri Net has a single sink, always the same, p190, and its output corresponds to the requirements output, then there is no complete shortfall.

PARTIAL SHORTFALLS

In every one of the twenty System Strategy Nets, there are always two partial shortfalls. These correspond to Pa6 and Pa8, in the case of the ten System Strategy Nets denoted from 1 to 10. The shortfalls for the other ten System Strategy Nets noted from 11 to 20 correspond to Pa6 and Pa9, where:

Pa9: $p2 \mid f2 \rightarrow f5\text{-}1 \rightarrow f6 \rightarrow f4 \rightarrow f7 \rightarrow f8 \rightarrow f10$.

For all twenty nets, there is a lack of use of some critical messages in the simple functionalities corresponding to Pa3, Pa4, and Pa5. Further, and this is due to the difference of strategy, in the case of the System Strategy Nets labeled from 1 to 10 there is also a lack of use of some critical data in the case of the simple functionalities corresponding to Pa7, while in the case of the System Strategy Nets labeled from 11 to 20, there is also a lack of use of some critical messages for the simple functionalities corresponding to Pa10, with:

Pa10: p2 | f2 $\rightarrow$ f5-1 $\rightarrow$ f6 $\rightarrow$ f4 $\rightarrow$ f7 $\rightarrow$ f9 $\rightarrow$ f10.

REDUNDANCY

There are no redundancies with conflict in any of the System Strategy Nets. Further, any redundancy is of the concurrency type.

COORDINATION

In the case of System Strategy Net 3, 8, 13 and 18, deadlock can occur because f10-1-1, and f10-1-2 do not receive all the input messages they should expect. However, this is not a major problem as there is a third f10 function, f10-1-3 which does not deadlock. However, in the case of these nets, the use of resources is far from optimum.

There are no other major coordination problems other than those resulting from partial shortfalls.

CONCLUSION

MESACC has certainly the potential to be a very endurable command center, due to its extensive redundancy with concurrency. However, as it is, MESACC does not fulfill its requirements. There are some partial shortfalls to be overcome. However, all the processes necessary for the implementation of these missing simple functionalities are in place, and some structural changes are possible to introduce these missing functionalities

CHAPTER VI

CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

6.1 CONCLUSIONS

The purpose of this study was to evaluate the operations of a proposed or implemented system against its required functionality. The framework of the research carried out is the structural analysis of compatible representations of the requirements and of the system with Petri Nets. The argument used can be expressed in five stages:

1) representation of the required system and of its proposed or implemented counterpart with Petri Nets for a given scenario.
2) establishment of correspondences between some structural properties of the Petri Net representation of the system and the functions it performs.
3) development of a set of tools to obtain the sub-nets of interest.
4) definition of various types of shortfalls and overlaps, and of tools to discriminate among them.
5) investigation of specific examples

The representation of the required system and of its proposed or implemented counterpart with Petri Nets allows to produce a compatible representation of the two entities. Petri Nets are ideal tools to represent both synchronization and concurrency. The modeling of the two entities is made with the use of switches that account for various types of variability, either to allow the system to adapt its processing to the inputs it receives, or to allow the system to adapt its structure to the environment.

When this is done, a scenario is chosen which reduces the size of the Petri Net. The use of a scenario is essential to the consistency of both representations.

On the structural side, a simple information flow path has been defined as a simple path from a source of the net to a sink, while a complete information flow path is a sub-net of the Petri Net that contains all the simple information flow paths that end at the same sink. The above definition holds whether the Petri Net contains cycles or not. Therefore, a simple information flow path traces one specific trajectory that a token can follow from a node to another node. A complete information flow path shows all the information and all the processing needed to generate a specific output by the system.

The term functionality has been used to describe a set of coordinated functions that a system must be capable of carrying out in order to accomplish a task or a sub-task. A simple functionality represents a sequence of processes that operates on an input message to produce an output. A simple functionality tracks the processing of a single input as it affects a single output. A complete functionality is the set of coordinated processes that operates on all the necessary inputs to produce an output.

The two sets of definitions have been constructed so that a correspondence exists between the structural properties of a Petri Net model of a system and the functions that this system performs. A simple information flow path, a structural element of a Petri Net, corresponds to a simple functionality, while a complete information flow path corresponds to a complete functionality.

Given this correspondence, the problem becomes one of developing algorithms for the determination of the simple and complete information flow paths. The theory of S-Invariants of ordinary Petri Net has been used to devise an algorithm to construct the subnets representing each complete functionality, while an enhanced version of a graph-theoretic algorithm has been used to determine the simple information flow paths. These constructs, obtained for both representations - the requirements net and the net that describes the proposed or implemented system - form the basis for the comparison and evaluation.

A complete shortfall has been defined as the existence of a complete functionality in the requirements that is not present in the system, while a partial shortfall has been defined as the existence of a simple functionality in the requirements net that is not present in the system. Here a distinction is made between a partial shortfall resulting from reduced processing by the system of all the appropriate inputs, or from not using all the inputs.

Three types of redundancies have been shown. First, a redundancy with conflict has been defined as the existence of two equivalent mutually exclusive simple functionalities with no particular embedded rule allowing resolution. Therefore, such redundancy is detrimental to the efficient operation of the system. Second, a redundancy with concurrency has been defined as the existence of two equivalent simple functionalities which produce similar outputs in the system. While more resources are used, both speed of response time and higher survivability and reliability may be achieved. Third, a redundancy with synchronization has been defined as the existence of two equivalent simple functionalities which pproduce outputs that need to be fused in order for the system to carry out its task. Although the quality of the output may improve as a result of the fusion, such a structure reduces reliability as the failure of any component in one of the simple functionalities jeopardizes the production of the output, and increases the response time.

Lastly, issues of coordination among simple functionalities have been tackled that can produce deadlocks, i.e., a function cannot be performed because it lacks one of its input.

The application of the methodology to MESACC, a hypothetical command center, has shown the complexity of such an evaluation. However, the graphical nature of the Petri Net, and the possibility to apply this methodology at an early stage of the conception, design and implementation of a distributed system allow a quick assessment of a design believed to be meeting some requirements, or the re-evaluation of an existing system versus new requirements resulting from evolution of the context in which the system operates.

6.2 DIRECTIONS FOR FURTHER RESEARCH

Research can be carried out in several directions to integrate the different concepts, models and evaluation tools presented in this work in a comprehensive study of the evaluation of functionality in distributed systems. Suggestions for future work are presented in the remainder of this section.

First, the SEA methodology could be applied to study the effect that individual shortfalls and overlaps have on the system. It would be interesting to study the behavior of a distributed system when various types of shortfalls and overlaps are introduced. For example, one can investigate the nature of the trade off between improved accuracy and lower response time in the case of redundancies with conflict, or the effect that the existence of a partial shortfall has on the output response of a system.

Second, it would be interesting to develop tools to evaluate such shortfalls and overlaps in the case of colored nets, thus allowing the investigation of the role that some protocols may play in the performance of the system.

Third, one could develop more efficient procedures for identifying and classifying the various types of shortfalls and overlaps. In particular, more sophisticated tools may be needed to evaluate structurally interleaved shortfalls and overlaps.

Fourth, there is an effort to be made to adapt the currently existing algorithms to construct complete and simple functionalities. As a matter of fact, the algorithm to construct the complete functionality tends to create dynamically huge data structures when implemented on a computer, as a result of some typical arrangement of the columns of the incidence matrix of the Petri Net. As a matter of fact, the first step of this algorithm involves the computation of a generator family of minimal support S-Invariants. This algorithm is described in detail in Appendix A. It appears in some occurrences, e.g., in the case of an incidence matrix with two hundred places and hundred and forty transitions, that several hundred new lines (representing new S-Invariants) are generated temprarily in an intermediate step of the process; most of them are eliminated by the end of the step. This consumes a lot of the internal memory of a computer, and leads to a memory overflow. A solution to that problem could be to create a more efficient heuristic procedure to limit the number of newly generated lines and test for the minimality of the support of each new vector, before generating new lines.

Finally, the development of a software package to integrate the various tools and to perform an automated evaluation of the functionality of a distributed system versus its requirements may be required in order to gain further insights in these types of problems.

REFERENCES

Alaiwan, H., and Toudic, J. M., 1985, "Recherche des Semi-Flots, des Verrous et des Trappes dans les Réseaux de Petri," *Technique et Science Informatiques*, Vol 4, No 1, Dunod, France, pp. 103-112.

Andreadakis, S. K., 1988, "Analysis and Synthesis of Decisionmaking Organizations," Ph.D Thesis, LIDS-TH-1740, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Beijjani, G., and Levis, A., 1985, "Information Storage and Access in Decisionmaking Organizations," *Proc. 8th MIT/ONR Workshop on C^3 Systems*, LIDS-R-1519, Laboratory for Information and Decision Systems, MIT, Cambridge MA.

BRAMS, G. W., 1983, *Réseaux de Petri: Théorie et Pratique tome 1: Théorie et Analyse*, Masson, Paris, France.

BRAMS, G. W., 1983, *Réseaux de Petri: Théorie et Pratique tome 2: Modélisation et Applications*, Masson, Paris, France.

Bouthonnier, V., and A. H. Levis, 1984, "Effectiveness Analysis for C^3 Systems." *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-14.

Cothier, P.H., and A.H.Levis, 1986, "Timeliness and Measures of Effectiveness in Command and Control," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-16, No.6.

Demaël, J., 1989, "On the Generation of Variable Structure Distributed Architectures," S.M. Thesis, Laboratory for Information and Decision Systems, MIT, Cambridge, MA. (in preparation)

Dersin, P., and A.H. Levis, 1981, "Large Scale System Effectiveness Analysis," LIDS-FR-1072, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Genrich, H. 1987, "Predicate Transition Nets," in *Advance Course on Petri Nets 1986*, in Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany.

Genrich, H. and Lautenbach, K., 1981, "System Modeling with High-Level Petri Nets," in *Theoretical Computer Science*, No. 13, pp. 109-136.

Grevet, J.-L., 1988, "Decision Aiding and Coordination in Decisionmaking Organizations," LIDS-TH-1737, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Jensen, K., 1987, "Colored Petri Nets" in *Advanced Course on Petri Nets 1986*, Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany.

Jin, V. Y., A. H. Levis and P. Remy, 1986, "Delays in Acyclical Distributed Decisionmaking Organizations," *4th IFAC Symposium on Large Scale Systems: Theory and Applications*, Zurich, Switzerland.

Karam, J.-G., 1985, "Effectiveness Analysis of Evolving Systems," LIDS-TH-1431, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Martin, P., 1986, "Large Scale C^3 Systems: Experiment, Design, And System Improvement," LIDS-TH-1580, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Martinez, J., and Silva, M., 1980, "A simple and fast algorithm to obtain all invariants of a generalised Petri Net,", in *Lecture Notes in Computer Science*, No. 52, Spinger Verlag, FRG, pp. 302-310.

Memmi, G., and Roucairol, G., 1979, "Linear Algebra in Net Theory," in *Net Theory and Application*, Lecture Notes in Computer Science, No. 84, Spinger Verlag, FRG, pp. 213-223.

Monguillet, J.-M., 1987, "Modeling and Evaluation of Variable Structure Organizations," LIDS-TH-1730, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Perdu, D., and A. H. Levis, 1989, "Evaluation of Expert System in Decisionmaking Organizations" in *Science of Command and Control, Coping with Uncertainty*, Edited by S. E. Johnson and A. H. Levis, AFCEA International Press, Wahington D.C.

Peterson, J. L., 1981, *Petri Net theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs, NJ.

Remy, P., and A. H. Levis, 1988, "On the Generation Of Organizational Architectures Using Petri Nets," in *Advances in Petri Nets 1988*, Lecture Notes in Computer Science, Springer Verlag, Berlin.

Reisig, W., 1985, *Petri Net, An Introduction*, Springer Verlag, Berlin.

Signori, D., and S. H. Starr, 1987, "The Mission Oriented Approach to Nato C^2 Planning," *Signal*, Vol. 42, No. 1, September.

Sweet, R., 1986, "An evolving C^2 Evaluation Tool - MCES Theory," *Proc. 9th MIT/ONR Workshop on C^3 Systems*. LIDS-R-1624, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Tabak, D., and A. H. Levis, 1985, "Petri Net Representation of Decision Models," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-15, No 6, pp. 812-818.

Weingaertner, S. T., and A. H. Levis, 1989, "Evaluation of Decision Aiding in Submarine Emergency Decisionmaking," *Automatica*, Vol. 25, No 3.

APPENDIX A

ALGORITHM TO OBTAIN ALL THE MINIMAL SUPPORT S-INVARIANTS OF AN ORDINARY PETRI NET

The complete proof of the algorithm can be found in Alaiwan and Toudic (1985). The description of the algorithm is presented first, and then an explanation of how it works is given.

In the following desription, C is assumed to be the Incidence Matrix of the Petri Net, of dimension $n \times m$, where n is the number of places and m the number of transitions. C_{ij} denotes, as usual, the elements corresponding to the i-th row and j-th column of C and I_n denotes the $n \times n$ identity matrix.

The objective of the algorithm is to find a family of S-Invariants all having a minimal support relative to the set of S-Invariants. Therefore, a minimal generator family will be obtained, and any S-Invariant will be obntained as a linear combination with **rational** coefficients of the elements of this family of S-Invariants. If it is imposed that the greatest common divisor of all the values of the non null elements of any minimal support S-Invariant be one, then there is unicity of such a generator family. Although this family is a generator family and is minimal, it is not a basis, as any linear combination of these S-Invariants is not necessarily an S-Invariant.

ALGORITHM:

```
BEGIN
    $A = C$ ; $D = I_n$                      { initialization, the matrix [ $I_n$ , C] is considered }
    repeat for j = 1                          { main loop }
        Determine the sets $I_1$ and $I_2$ such that:
            $I = \{i$ such that $a_{ij} > 0\}$ and $K = \{k$ such that $A_{kj} < 0\}$
        For all pairs $(i, k) \in I \times K$ do
            Append to the Matrix $[D_j , A_j] = E_j$ the row vector:
                $E_{ij}$ * (kth row of E) - $E_{kj}$ * (ith row of E) { the resulting jth element is null }
        end
        Eliminate the jth column of $A_j$
        Eliminate from $E_j$ all the rows with index $i \in I \cup K$
        Eliminate from $E_j$ all the rows, which when restricted to $D_j$ have a non-minimal support
        with respect to the other rows of D. If two rows have the same support, eliminate one of
        them.
    until j = m
    The rows of $D_m$ determine all the minimal support S-invariants of the net.
END.
```

Proof: By induction

Initially, all vectors of $D = I_n$ contains all the minimal support S-Invariants of all its transitions (we denote by R^i the Petri Net obtained by eliminating the transitions $\{i+1, ..., m\}$). Starting from the minimal support S-Invariants of R_i, the algorithm will generate all R_{i+1} S-Invariants. Any R_{i+1} S-Invariant, Y^j_{i+1}, is also an R_i S-Invariant, although it is not necessarily a minimal support S-Invariant. Therefore, any Y^j_{i+1} can be written as a linear combination of the R_i minimal support S-Invariants.

EXAMPLE: Consider the net depicted in Figure A.1.

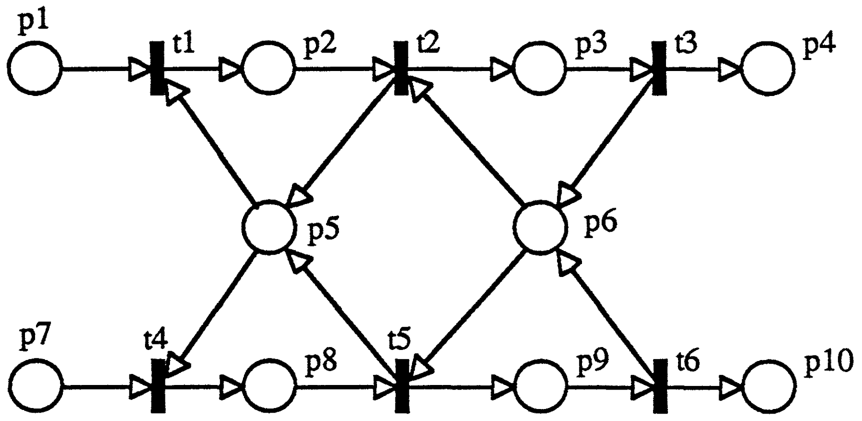


Figure A.1 Petri Net PNa

The computation starts with matrix E_1, provided below:

$$\begin{array}{lcccccc}
1000000000 & -1 & 0 & 0 & 0 & 0 & 0 \\
0100000000 & 1 & -1 & 0 & 0 & 0 & 0 \\
0010000000 & 0 & 1 & -1 & 0 & 0 & 0 \\
0001000000 & 0 & 0 & 1 & 0 & 0 & 0 \\
0000100000 & -1 & 1 & 0 & -1 & 1 & 0 \\
0000010000 & 0 & -1 & 1 & 0 & -1 & 1 \\
0000001000 & 0 & 0 & 0 & -1 & 0 & 0 \\
0000000100 & 0 & 0 & 0 & 1 & -1 & 0 \\
0000000010 & 0 & 0 & 0 & 0 & 1 & -1 \\
0000000001 & 0 & 0 & 0 & 0 & 0 & 1 \\
I_n & & & C & & &
\end{array}$$

I_n represents all the minimal support S-Invariants of R_0, (each place is the support of a minimal support S-Invariant).

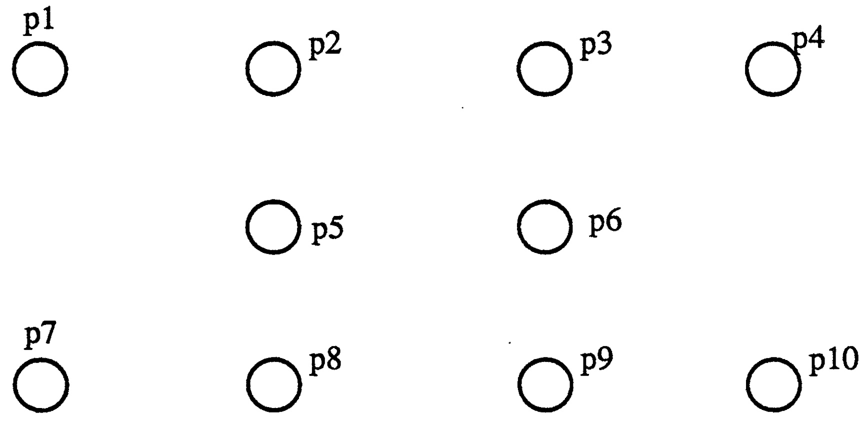


Figure A.2 Petri Net R_0

By applying the algorithm, E_2 is constructed, and presented below:

~~1 0 0 0 0 0 0 0 0 0~~	~~1 0 0 0 0 0~~	
~~0 1 0 0 0 0 0 0 0 0~~	~~1 -1 0 0 0 0~~	
0 0 1 0 0 0 0 0 0 0	0 1 -1 0 0 0	
0 0 0 1 0 0 0 0 0 0	0 0 1 0 0 0	
~~0 0 0 0 1 0 0 0 0 0~~	~~-1 1 0 -1 1 0~~	
0 0 0 0 0 1 0 0 0 0	0 -1 1 0 -1 1	
0 0 0 0 0 0 1 0 0 0	0 0 0 -1 0 0	
0 0 0 0 0 0 0 1 0 0	0 0 0 1 -1 0	
0 0 0 0 0 0 0 0 1 0	0 0 0 0 1 -1	
0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 1	
1 1 0 0 0 0 0 0 0 0	0 -1 0 0 0 0	New rows
0 1 0 0 1 0 0 0 0 0	0 0 0 -1 1 0	

It is still possible to check that the rows of D_1 are the minimal support S-Invariants of the Petri Net R_1.

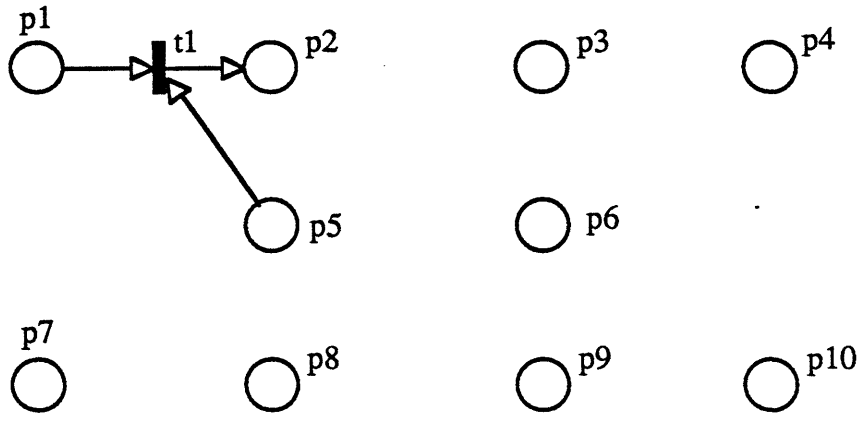


Figure A.3 Petri Net R_1

In this appendix, the main loop is applied a last time, and thus E_3 is constructed, and presented below:

~~1 0 0 0 0 0 0 0 0 0~~	~~1~~	~~0 0 0 0 0~~	
~~0 1 0 0 0 0 0 0 0 0~~	~~1~~	~~1 0 0 0 0~~	
~~0 0 1 0 0 0 0 0 0 0~~	~~0~~	~~1 -1 0 0 0~~	
0 0 0 1 0 0 0 0 0 0	0	0 1 0 0 0	
~~0 0 0 0 1 0 0 0 0 0~~	~~1~~	~~1 0 1 1 0~~	
~~0 0 0 0 0 1 0 0 0 0~~	~~0~~	~~-1 1 0 -1 1~~	
0 0 0 0 0 0 1 0 0 0	0	0 0 -1 0 0	
0 0 0 0 0 0 0 1 0 0	0	0 0 1 -1 0	
0 0 0 0 0 0 0 0 1 0	0	0 0 0 1 -1	
0 0 0 0 0 0 0 0 0 1	0	0 0 0 0 1	
~~1 1 0 0 0 0 0 0 0 0~~	~~0~~	~~-1 0 0 0 0~~	
0 1 0 0 1 0 0 0 0 0	0	0 0 -1 1 0	
0 0 1 0 0 1 0 0 0 0		0 0 0 -1 1	New rows
1 1 1 0 0 0 0 0 0 0		0 -1 0 0 0	

It is still possible to verify that the rows of D_2 are the minimal support S-Invariants of the Petri Net R_2.

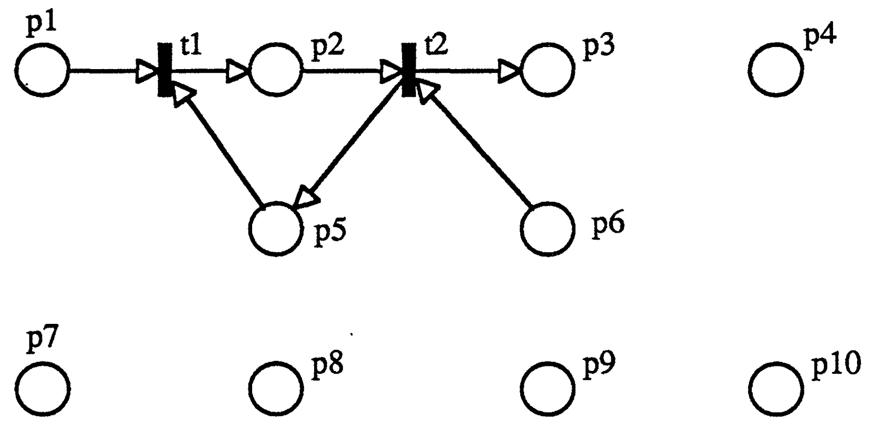


Figure A.4 Petri Net R_2

CONCLUSION: This is a fast (incidence matrices are in general shallow) and powerful algorithm, which generates a generator family of minimal support S-Invariants.

APPENDIX B

PROPERTIES OF S-INVARIANTS IN A DECISION FREE PETRI NET

Some properties of S-Invariants, defined in chapter II, are presented here. Some of them are general, while others are more specific to the class of Petri Nets considered in this work. The class of Petri Nets used has the following characteristics:

- The Petri Net contains a set of sources, i.e., a set of places with no input transitions. Sources represent nodes where information arrives from the sensors of the system.
- The Petri Net contains a set of sinks, i.e., a set of places with no output transitions. Sinks represent outputs or responses of the system to the environment.
- The Petri Net is decision free, i.e., there is no place with more than one output transition. This property is translated into the following relation:

$$\forall\, p \in P, \quad \text{card}(p^{\cdot}) \leq 1 \qquad \text{(B.1)}$$

More precisely, if a place is not a sink, we have, for any other place of the net the following relation:

$$\forall\, p \in P, \quad \text{card}(p^{\cdot}) = 1 \qquad \text{(B.2)}$$

A Petri Net model of a distributed system may contain directed circuits. Among these directed circuits, two structurally different classes are distinguished. These two classes are presented below.

Definition B.1:

A *resource loop* is an elementary circuit which is a marked graph.

Definition B.2:

An *information loop* is a directed circuit such that it contains at least a place that has more than one input transition.

Property B.1:

Given the class of Petri Net used in this work, any loop belongs in one of the two categories.

Proof:

If a circuit is a marked graph, then, none of its places has more than one input transition. Conversely, a circuit that contains a place that has more than one input transition is not a marked graph. No loop can belong to both categories.

The decision free nature of the Petri Nets under consideration in this work insures that a directed circuit does not contain any place having more than one output transition. Therefore any loop, which is not a marked graph is such that it contains at least a place having more than one input transition. Thus, every loop belongs to one of the two categories.

In a Petri Net model of a distributed system, a resource loop is used to model the availability of a resource to a function or a group of functions. An information loop is used to model the storage of data or information. As a matter of fact, it will be shown later that a flow of information in a net that contains an information loop stops in the loop.

Lemma B.1:

In a Petri Net, the following conditions are **necessary** for a subnet to be an S-Component:

- i) there are no transitions in the net having their preset being empty.
- ii) there are no transitions in the net having their postset being empty.

Proof:

Given a subnet, if it is the S-Component of an S-Invariant, then, for the n places of the subnet, the following relation holds:

$$\sum_{i=1}^{n} M(p_i) \cdot x_i = \text{constant} \qquad \text{(B.3)}$$

where any x_i is strictly positive.

If i) is not verified, then there exists a transition which, when firing, removes a token from a place, p_n, of the subnet, and does not create any token in any place of the net. Therefore, after the firing of that transition, the left hand side of the relation becomes:

$$\sum_{i=1}^{n-1} M(p_i) \cdot x_i + (M(p_n) - 1) \cdot x_n \qquad \text{(B.4)}$$

The difference between the left hand side of relation (B.3), and relation (B.4) is x_n. By definition of an S-Invariant, x_n is non null, and therefore, relation (B.3) is not verified whatever the firing. The subnet cannot be the S-Component of an S-Invariant.

Similarly, if ii) is not verified, then there exists a transition which, when firing, creates a token in a place, p_n, of the subnet, and does not remove any token in any place of the subnet. Therefore, after the firing of that transition, the left hand side of the relation becomes:

$$\sum_{i=1}^{n-1} M(p_i) \cdot x_i + (M(p_n) + 1) \cdot x_n \qquad \text{(B.5)}$$

The difference between relation (B.5), and the left member of relation (B.3) is x_n. By definition of an S-Invariant, x_n is non null, and therefore, relation (B.3) is not verified whatever the firing. The subnet cannot be the S-Component of an S-Invariant.

Lemma B.2:

Given a subnet, a **necessary** condition for it to be an S-Component is that, for any place of the subnet, the number of input transitions in the subnet equals the number of its input transitions in the Net.

In other words, the necessary condition is that the subnet must contain all the input transitions of any of its places in the net.

Proof:

If it is supposed that a place p1 of an S-Component, S, is such that one of its input transitions, t1, is not present in the subnet, then by definition of an S-Component, no input place of t1 belongs to the subnet. Therefore, $\{\cdot t\} \cap P_S = \emptyset$. The firing of t1 creates a new token in p1 and no token is removed from any place of the subnet. Therefore, ii) from lemma B.1 can be applied. The subnet is not an S-Component.

Lemma B.3:

In a Petri Net, if a circuit is a marked graph, then this circuit is a minimal support S-Invariant.

This lemma states that a resource loop is a minimal support S-Invariant. This result has been generalized by Hillion, (1986), who proved that in a Petri Net, any directed elementary circuit is a minimal support S-Invariant.

Proof:

Given that the circuit is a marked graph, the firing of any transition in the circuit removes a token from its input place that belongs to the circuit, and creates a token in the output place that belongs to the circuit. Therefore, there is conservation of tokens in the circuit. If the firing of a transition that does not belong to the circuit is considered, then no token can be removed from a place of the circuit, since by hypothesis it is a marked graph, and as such, each place has only one output transition that is part of the circuit. Similarly, no token can be created in the circuit since each place of the circuit has only one input transition that is part of the circuit. The S-Invariant is a minimal support S-Invariant since the removal of places in the circuit creates a subnet that is a simple path. In this path, there is at least a place such that its input transition, t1, does not belong to the path. In turn, this place has no input place belonging to the path. As a result, the firing of t1 allows the creation of a token in a place without any token removal from any place of the path. Therefore, ii) from lemma B.1 can be applied. The subnet is not an S-Component.

Lemma B.4:

In a Petri Net with sources and sinks, given a subnet which is not a resource loop, if it does not contain a source, then it is not an S-Component.

Proof: Consider an S-Component which is not a resource loop, and one of its places, p_i. If this place is not a source, then, rule ii) of lemma B.1 states that there exists a transition, t_i, belonging to $\cdot p_i$, the non empty preset of p_i. Rule i) of lemma B.1 can in turn be applied to t_i. Therefore, there exists a place, p_j, belonging to the non empty preset of t_i. This process can be applied recursively to construct a sequence of nodes. As the subnet is not a resource loop, it is possible to construct that sequence of nodes without including twice the same node in the sequence. Therefore, given the finite nature of the net and the constraint of lemma B.1, the sequence stops when a source is encountered.

Lemma B.5:

A directed circuit, in which at least one of its places, p_i, has more than one input transition is not an S-Component. More precisely, any S-Invariant whose S-Component contains such a loop, contains at least a directed path from a source to the place p_i. Further, to be the S-Component of a **minimal support** S-Invariant, such S-component cannot contain 1) any directed path originating from a node of the loop and containing nodes that do not belong to the loop, and 2) any directed path from a source to the loop, whose first node belongs to the loop is a transition.

This lemma states that:

a) an information loop can not be a minimal support S-Invariant.

b) any S-component that contains an information loop must contain some directed path from sources to the multi input places of the loop.

c) any S-Component of a minimal support S-Invariant that contains an information loop, does not contain any directed path originating from a node of the information loop and containing nodes that do not belong to the loop.

d) any S-Component of a minimal support S-Invariant that contains an information loop does not contain any directed path from a source to the information loop whose first node belonging to the loop is a transition.

Proof:

Because p_i has an input transition that does not belong to the information loop, the firing of that transition creates a new token in p_i, but does not affect the marking of the other places of the directed circuit. Therefore the result of ii) of lemma B.1 is applicable: an information loop is not the S-Component of an S-Invariant.

Lemma B.3 can also be applied to the information loop, and therefore, if there is not at least a directed path from a source to p_i that is part of the subnet containing the information loop, then that subnet cannot be an S-Invariant.

Given the decision free nature of the Petri Net, any directed path originating from a node of the loop contains a transition, t0, of the loop and one of its output place, p0, that does not belong to the loop itself. If this directed path is part of the S-Component of a minimal support S-Invariant which also contains the information loop, then a relation similar to (B.3) exists. The firing sequence involving the directed sequence of transitions of the loop, starting from t0 is considered. The result of that firing sequence is that the marking of the loop is unchanged, and a token has been created in p0. Therefore, the weighted conservation of token can hold only if, when t0 fires, a token is removed from an input place to t0 that is not part of the loop. In this case, the firing of t0 results in a twofold independent conservation of tokens, one within the loop, and the other along the directed path which contains only one node of the loop, t0. In this case, there is an S-Invariant which is the result of the combination of at least two minimal support S-Invariants. That subnet is not the S-component of a minimal support S-Invariant.

Similarly, if a directed path existed in the S-Component whose first node in common with the loop in its sequence is a transition, t_i, the same firing sequence as above would show the creation of a token in the output place of t_i that belongs to the loop, p_i, and the marking of the other places of the loop would be unchanged. Therefore, the weighted conservation of token can hold only if, when t_i fires, a token is created in an output place of t_i that is not part of the loop. In this case, the firing of t_i results in a twofold independent conservation of tokens, one within the loop, and the other along the directed path which contains only one node of the loop, t_i. In this case, there is an S-Invariant which is the result of the combination of at least two minimal support S-Invariants. That subnet is not the S-component of a minimal support S-Invariant.

Corollary B.1:

A subnet which is the S-Component of a minimal support S-Invariant and which contains an information loop does not contain any sink.

Proof:

If such a subnet existed, then there would be a directed path from a node of the loop to the sink. This is in contradiction with the result of lemma B.3 that in order to be a minimal support S-Invariant, an S-Component containing an information loop does not contain such a directed path.

Lemma B.6:

In a decision free Petri Net, there is no subnet that contains two sinks and which is the S-Component of a **minimal support** S-Invariant.

Proof:

Suppose there is a subnet with two sinks, and it is a minimal support S-Invariant. Then, according to corollary B.1, this subnet contains sources and it can be inferred that it consists of a set of directed paths from sources to sinks. Furthermore, as the subnet is, by hypothesis, a minimal support S-Invariant, then there is no loop attached to it. If there exists more than one sink, then that means that some directed paths diverge at a certain node. This node cannot be a place as it would contradict the decision free assumption. Therefore the node is a transition. If the node is a transition, then it is clear that there exists a conservation of tokens for the set of directed paths from the sources to a given sink, regardless of what the firing is of the other transitions in the other branch, from the transition to the sink. Therefore, we can evidence another S-Invariant whose support is included in the support of the other S-Invariant. Therefore, the subnet cannot be the S-Component of a minimal support S-Invariant.

Theorem B.1:

In a **decision free** Petri Net with **multiple sources** and **multiple sinks**, there are at most three classes of **minimal support** S-Invariants:

a) minimal support S-Invariants whose S-Components are resource loop.
b) minimal support S-Invariants whose S-Components are sets of directed paths from a set of sources to a set of sinks, and which do not include any loop.
c) minimal support S-Invariants whose support is made of:
   - a set of directed paths from a set of sources to an information loop.
   - the information loop itself.

Proof:

In lemma B.3, it has been established that a resource loop is the S-Component of a minimal support S-Invariant. Therefore, any subnet containing a resource loop cannot be the S-Component of a minimal support S-Invariant, as the support of the resource loop would be included in the support of the subnet.

Suppose there exists an S-Component of a minimal support S-Invariant which consists of directed paths from a set of sources to a set of sinks, and an information loop. It has been established in corollary B.1 that a minimal support S-Invariant whose support contains an information loop cannot contain any sink. Therefore, the S-Component under consideration cannot be inferred from a minimal support S-Invariant.

In this appendix, minimal support S-Invariants have been categorized into three classes, as described in theorem B.1. Further, the decision free nature of the Petri Nets implies that there is no minimal support S-Invariant containing in its support more than one sink.

The methodology to construct complete information flow paths consists of two steps after the S-Components have been determined. In the first step, the S-Components that contain the sink of the complete information flow path are coalesced together. Theorem B.1 shows that these S-Components are of type b). In the second step, the S-Components added are of type a) and c), and correspond to S-Components containing a cycle.

APPENDIX C

MINIMAL SUPPORT S-INVARIANTS IN THE SYSTEM NET # 1

In this appendix, the full list of all the minimal support S-Invariants that are present in the stategy system net 1 is presented. They have been obtained by a computer implementation of the algorithm by Alaiwan and Toudic (1985) presented in Appendix A. The listing of these minimal support S-Invariants is made by providing the support of each S-Invariant.

Invariant 1:
c2 c3 c4 c5 p1 p14 p35 p42 p49
Invariant 2:
c2 c3 c4 c5 p1 p14 p37 p44 p51
Invariant 3:
c11 c12 p62 p63 j5 p67 p69 p71
Invariant 4:
c10 c11 c12 p61 j3 p110 j11 p116 p117 p122
Invariant 5:
c2 c3 c4 c5 p1 p14 p31 j3 p110 j11 p116 p117 p122
Invariant 6:
c10 c11 c12 p61 j3 p111 p112 j11 p116 p117 p122
Invariant 7:
c2 c3 c4 c5 p1 p14 p31 j3 p111 p112 j11 p116 p117 p122
Invariant 8:
c2 c3 c4 c5 p1 p14 p33 p177 p187 p188 j8-1 p190
Invariant 9:
c10 c11 c12 c13 p3 p15 p53 p149 p177 p187 p188 j7-2 j8-1 p190
Invariant 10:
c10 c11 c12 c13 p3 p15 p53 p147 p148 p177 p187 p188 j7-2 j8-1 p190
Invariant 11:
c12 p74 p75 p76 j4 p177 p182 p187 p188 j9-1 p190

Invariant 12:

c2 c3 c4 c5 c12 p1 p14 p36 p75 p76 j4 p177 p182 p187 p188
j9-1 p190

Invariant 13:

c11 c12 p62 p63 j5 p68 p75 p76 j4 p177 p182 p187 p188 j9-1
p190

Invariant 14:

c12 p74 p75 p78 p82 j4 p177 p182 p187 p188 j9-1 p190

Invariant 15:

c2 c3 c4 c5 c12 p1 p14 p36 p75 p78 p82 j4 p177 p182 p187
p188 j9-1 p190

Invariant 16:

c11 c12 p62 p63 j5 p68 p75 p78 p82 j4 p177 p182 p187 p188 j9-1 p190

Invariant 17:

c12 p74 p75 p79 p83 j4 p177 p182 p187 p188 j9-1 p190

Invariant 18:

c2 c3 c4 c5 c12 p1 p14 p36 p75 p79 p83 j4 p177 p182 p187 p188 j9-1 p190

Invariant 19:

c11 c12 p62 p63 j5 p68 p75 p79 p83 j4 p177 p182 p187 p188 j9-1 p190

Invariant 20:

c2 c3 c4 c5 p1 p14 p33 p178 p179 p187 p188 j8-1 p190

Invariant 21:

c10 c11 c12 c13 p3 p15 p53 p149 p178 p179 p187 p188 j7-2 j8-1 p190

Invariant 22:

c10 c11 c12 c13 p3 p15 p53 p147 p148 p178 p179 p187 p188 j7-2 j8-1 p190

Invariant 23:

c12 p74 p75 p76 j4 p178 p179 p182 p187 p188 j9-1 p190

Invariant 24:

c2 c3 c4 c5 c12 p1 p14 p36 p75 p76 j4 p178 p179 p182 p187 p188 j9-1 p190

Invariant 25:

c11 c12 p62 p63 j5 p68 p75 p76 j4 p178 p179 p182 p187 p188 j9-1 p190

Invariant 26:

c12 p74 p75 p78 p82 j4 p178 p179 p182 p187 p188 j9-1 p190

Invariant 27:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p78 p82 j4 p178 p179 p182 p187 p188 j9-1 p190
Invariant 28:
c11 c12 p62 p63 j5 p68 p75 p78 p82 j4 p178 p179 p182 p187 p188 j9-1 p190
Invariant 29:
c12 p74 p75 p79 p83 j4 p178 p179 p182 p187 p188 j9-1 p190
Invariant 30:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p79 p83 j4 p178 p179 p182 p187 p188 j9-1 p190
Invariant 31:
c11 c12 p62 p63 j5 p68 p75 p79 p83 j4 p178 p179 p182 p187 p188 j9-1 p190
Invariant 32:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p33 p61 j3 p110 j11 p114 p177 p187 j8-1 p190
Invariant 33:
c10 c11 c12 c13 p3 p15 p53 p61 j3 p110 j11 p114 p149 p177 p187 j7-2 j8-1 p190
Invariant 34:
c10 c11 c12 c13 p3 p15 p53 p61 j3 p110 j11 p114 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 35:
c10 c11 c12 p61 p74 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 36:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 37:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 38:
c10 c11 c12 p61 p74 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 39:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 40:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190

Invariant 41:
c10 c11 c12 p61 p74 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 42:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 43:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 44:
c2 c3 c4 c5 p1 p14 p31 p33 j3 p110 j11 p114 p177 p187 j8-1 p190
Invariant 45:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p110 j11 p114 p149 p177 p187 j7-2 j8-1 p190
Invariant 46:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p110 j11 p114 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 47:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 48:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 49:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 50:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 51:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 52:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190

Invariant 53:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 54:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 55:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 j9-1 p190
Invariant 56:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p33 p61 j3 p111 p112 j11 p114 p177 p187 j8-1 p190
Invariant 57:
c10 c11 c12 c13 p3 p15 p53 p61 j3 p111 p112 j11 p114 p149 p177 p187 j7-2 j8-1 p190
Invariant 58:
c10 c11 c12 c13 p3 p15 p53 p61 j3 p111 p112 j11 p114 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 59:
c10 c11 c12 p61 p74 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 60:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 61:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 62:
c10 c11 c12 p61 p74 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 63:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 64:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 65:
c10 c11 c12 p61 p74 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 66:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 67:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 68:
c2 c3 c4 c5 p1 p14 p31 p33 j3 p111 p112 j11 p114 p177 p187 j8-1 p190
Invariant 69:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p111 p112 j11 p114 p149 p177 p187 j7-2 j8-1 p190
Invariant 70:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p111 p112 j11 p114 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 71:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 72:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 73:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190
Invariant 74:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 75:

c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 76:

c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 77:

c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 78:

c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 79:

c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 j9-1 p190

Invariant 80:

c2 c3 c4 c5 c10 c11 c12 p1 p14 p33 p61 j3 p110 j11 p114 p178 p179 p187 j8-1 p190

Invariant 81:

c10 c11 c12 c13 p3 p15 p53 p61 j3 p110 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190

Invariant 82:

c10 c11 c12 c13 p3 p15 p53 p61 j3 p110 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190

Invariant 83:

c10 c11 c12 p61 p74 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190

Invariant 84:

c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190

Invariant 85:

c10 c11 c12 p61 p62 p63 j5 p68 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190

Invariant 86:
c10 c11 c12 p61 p74 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 87:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 88:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 89:
c10 c11 c12 p61 p74 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 90:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 91:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 92:
c2 c3 c4 c5 p1 p14 p31 p33 j3 p110 j11 p114 p178 p179 p187 j8-1 p190
Invariant 93:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p110 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 94:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p110 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 95:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 96:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190

Invariant 97:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 98:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 99:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 100:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 101:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 102:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 103:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 104:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p33 p61 j3 p111 p112 j11 p114 p178 p179 p187 j8-1 p190
Invariant 105:
c10 c11 c12 c13 p3 p15 p53 p61 j3 p111 p112 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 106:
c10 c11 c12 c13 p3 p15 p53 p61 j3 p111 p112 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 107:
c10 c11 c12 p61 p74 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190

Invariant 108:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 109:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 110:
c10 c11 c12 p61 p74 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 111:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 112:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 113:
c10 c11 c12 p61 p74 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 114:
c2 c3 c4 c5 c10 c11 c12 p1 p14 p36 p61 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 115:
c10 c11 c12 p61 p62 p63 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 116:
c2 c3 c4 c5 p1 p14 p31 p33 j3 p111 p112 j11 p114 p178 p179 p187 j8-1 p190
Invariant 117:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p111 p112 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 118:
c2 c3 c4 c5 c10 c11 c12 c13 p1 p3 p14 p15 p31 p53 j3 p111 p112 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190

Invariant 119:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 120:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 121:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 122:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 123:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 124:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 125:
c2 c3 c4 c5 c12 p1 p14 p31 p74 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 126:
c2 c3 c4 c5 c12 p1 p14 p31 p36 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 127:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p63 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 j9-1 p190
Invariant 128:
c-1 p95 p96 j1 p98 p104 p190
Invariant 129:
c-1 p95 p96 j1 p107 p108 p109
Invariant 130:
c10 c11 c12 p61 c-1 p95 p96 j1 p98 j3 p110 j11 p114 p190

Invariant 131:
c2 c3 c4 c5 p1 p14 p31 c-1 p95 p96 j1 p98 j3 p110 j11 p114 p190
Invariant 132:
c10 c11 c12 p61 c-1 p95 p96 j1 p98 j3 p111 p112 j11 p114 p190
Invariant 133:
c2 c3 c4 c5 p1 p14 p31 c-1 p95 p96 j1 p98 j3 p111 p112 j11 p114 p190
Invariant 134:
c2 c3 c4 c5 p1 p14 p33 c-1 p95 p96 j1 p98 p188 j8-1 p190
Invariant 135:
c10 c11 c12 c13 p3 p15 p53 c-1 p95 p96 j1 p98 p149 p188 j7-2 j8-1 p190
Invariant 136:
c10 c11 c12 c13 p3 p15 p53 c-1 p95 p96 j1 p98 p147 p148 p188 j7-2 j8-1 p190
Invariant 137:
c12 p74 p75 p76 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 138:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p76 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 139:
c11 c12 p62 p63 j5 p68 p75 p76 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 140:
c12 p74 p75 p78 p82 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 141:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p78 p82 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 142:
c11 c12 p62 p63 j5 p68 p75 p78 p82 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 143:
c12 p74 p75 p79 p83 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 144:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p79 p83 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 145:
c11 c12 p62 p63 j5 p68 p75 p79 p83 j4 c-1 p95 p96 j1 p98 p182 p188 j9-1 p190
Invariant 146:
c2 c3 c4 c5 p1 p14 p33 c-1 p95 p96 j1 p104 p177 p187 j8-1 p190

Invariant 147:
c10 c11 c12 c13 p3 p15 p53 c-1 p95 p96 j1 p104 p149 p177 p187 j7-2 j8-1 p190
Invariant 148:
c10 c11 c12 c13 p3 p15 p53 c-1 p95 p96 j1 p104 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 149:
c12 p74 p75 p76 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 150:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p76 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 151:
c11 c12 p62 p63 j5 p68 p75 p76 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 152:
c12 p74 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 153:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 154:
c11 c12 p62 p63 j5 p68 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 155:
c12 p74 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 156:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 157:
c11 c12 p62 p63 j5 p68 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p177 p182 p187 j9-1 p190
Invariant 158:
c2 c3 c4 c5 p1 p14 p33 c-1 p95 p96 j1 p104 p178 p179 p187 j8-1 p190
Invariant 159:
c10 c11 c12 c13 p3 p15 p53 c-1 p95 p96 j1 p104 p149 p178 p179 p187 j7-2 j8-1 p190

Invariant 160:
c10 c11 c12 c13 p3 p15 p53 c-1 p95 p96 j1 p104 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 161:
c12 p74 p75 p76 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 162:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p76 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 163:
c11 c12 p62 p63 j5 p68 p75 p76 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 164:
c12 p74 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 165:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 166:
c11 c12 p62 p63 j5 p68 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 167:
c12 p74 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 168:
c2 c3 c4 c5 c12 p1 p14 p36 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 169:
c11 c12 p62 p63 j5 p68 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 j9-1 p190
Invariant 170:
c6 c7 c8 c9 p2 p7 p13 p17 p22 p27
Invariant 171:
c6 c7 c8 c9 p2 p7 p13 p19 p149 p177 p187 p188 j7-2 j8-1 p190
Invariant 172:
c6 c7 c8 c9 p2 p7 p13 p19 p147 p148 p177 p187 p188 j7-2 j8-1 p190

Invariant 173:
c6 c7 c8 c9 p2 p7 p13 p19 p149 p178 p179 p187 p188 j7-2
j8-1 p190
Invariant 174:
c6 c7 c8 c9 p2 p7 p13 p19 p147 p148 p178 p179 p187 p188 j7-2 j8-1 p190
Invariant 175:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p110 j11 p114 p149 p177 p187 j7-2
j8-1 p190
Invariant 176:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p110 j11 p114 p147 p148 p177 p187
j7-2 j8-1 p190
Invariant 177:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p110 j11 p114 p149 p177
p187 j7-2 j8-1 p190
Invariant 178:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p110 j11 p114 p147 p148
p177 p187 j7-2 j8-1 p190
Invariant 179:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p111 p112 j11 p114 p149 p177 p187
j7-2 j8-1 p190
Invariant 180:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p111 p112 j11 p114 p147 p148 p177
p187 j7-2 j8-1 p190
Invariant 181:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p111 p112 j11 p114 p149
p177 p187 j7-2 j8-1 p190
Invariant 182:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p111 p112 j11 p114 p147
p148 p177 p187 j7-2 j8-1 p190
Invariant 183:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p110 j11 p114 p149 p178 p179 p187
j7-2 j8-1 p190

Invariant 184:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p110 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 185:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p110 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 186:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p110 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 187:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p111 p112 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 188:
c6 c7 c8 c9 c10 c11 c12 p2 p7 p13 p19 p61 j3 p111 p112 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 189:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p111 p112 j11 p114 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 190:
c6 c7 c8 c9 c2 c3 c4 c5 p1 p2 p7 p13 p14 p19 p31 j3 p111 p112 j11 p114 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 191:
c6 c7 c8 c9 p2 p7 p13 p19 c-1 p95 p96 j1 p98 p149 p188 j7-2 j8-1 p190
Invariant 192:
c6 c7 c8 c9 p2 p7 p13 p19 c-1 p95 p96 j1 p98 p147 p148 p188 j7-2 j8-1 p190
Invariant 193:
c6 c7 c8 c9 p2 p7 p13 p19 c-1 p95 p96 j1 p104 p149 p177 p187 j7-2 j8-1 p190
Invariant 194:
c6 c7 c8 c9 p2 p7 p13 p19 c-1 p95 p96 j1 p104 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 195:
c6 c7 c8 c9 p2 p7 p13 p19 c-1 p95 p96 j1 p104 p149 p178 p179 p187 j7-2 j8-1 p190

Invariant 196:
c6 c7 c8 c9 p2 p7 p13 p19 c-1 p95 p96 j1 p104 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 197:
c11 c12 p62 p64 p65 j5 p67 p69 p71 p191
Invariant 198:
c11 c12 p62 p64 p65 j5 p68 p75 p76 j4 p177 p182 p187 p188 p191 j9-1 p190
Invariant 199:
c11 c12 p62 p64 p65 j5 p68 p75 p78 p82 j4 p177 p182 p187 p188 p191 j9-1 p190
Invariant 200:
c11 c12 p62 p64 p65 j5 p68 p75 p79 p83 j4 p177 p182 p187 p188 p191 j9-1 p190
Invariant 201:
c11 c12 p62 p64 p65 j5 p68 p75 p76 j4 p178 p179 p182 p187 p188 p191 j9-1 p190
Invariant 202:
c11 c12 p62 p64 p65 j5 p68 p75 p78 p82 j4 p178 p179 p182 p187 p188 p191 j9-1 p190
Invariant 203:
c11 c12 p62 p64 p65 j5 p68 p75 p79 p83 j4 p178 p179 p182 p187 p188 p191 j9-1 p190
Invariant 204:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 205:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 206:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 207:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p76 j4 j3 p110 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 208:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p177 p182 p187 p191 j9-1 p190

Invariant 209:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 210:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 211:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 212:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 213:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 214:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 215:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p177 p182 p187 p191 j9-1 p190
Invariant 216:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 217:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 218:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 219:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p76 j4 j3 p110 j11 p114 p178 p179 p182 p187 p191 j9-1 p190

Invariant 220:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p110 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 221:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p110 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 222:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 223:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 224:
c10 c11 c12 p61 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 225:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p76 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 226:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p78 p82 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 227:
c2 c3 c4 c5 c11 c12 p1 p14 p31 p62 p64 p65 j5 p68 p75 p79 p83 j4 j3 p111 p112 j11 p114 p178 p179 p182 p187 p191 j9-1 p190
Invariant 228:
c11 c12 p62 p64 p65 j5 p68 p75 p76 j4 c-1 p95 p96 j1 p98 p182 p188 p191 j9-1 p190
Invariant 229:
c11 c12 p62 p64 p65 j5 p68 p75 p78 p82 j4 c-1 p95 p96 j1 p98 p182 p188 p191 j9-1 p190
Invariant 230:
c11 c12 p62 p64 p65 j5 p68 p75 p79 p83 j4 c-1 p95 p96 j1 p98 p182 p188 p191 j9-1 p190

Invariant 231:
c11 c12 p62 p64 p65 j5 p68 p75 p76 j4 c-1 p95 p96 j1 p104 p177 p182 p187 p191 j9-1 p190
Invariant 232:
c11 c12 p62 p64 p65 j5 p68 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p177 p182 p187 p191 j9-1 p190
Invariant 233:
c11 c12 p62 p64 p65 j5 p68 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p177 p182 p187 p191 j9-1 p190
Invariant 234:
c11 c12 p62 p64 p65 j5 p68 p75 p76 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 p191 j9-1 p190
Invariant 235:
c11 c12 p62 p64 p65 j5 p68 p75 p78 p82 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 p191 j9-1 p190
Invariant 236:
c11 c12 p62 p64 p65 j5 p68 p75 p79 p83 j4 c-1 p95 p96 j1 p104 p178 p179 p182 p187 p191 j9-1 p190
Invariant 237:
c-14 p94 p124 p127 p149 p177 p187 p188 j7-2 j8-1 p190
Invariant 238:
c-14 p94 p124 p127 p147 p148 p177 p187 p188 j7-2 j8-1 p190
Invariant 239:
c-14 p94 p124 p177 p187 p188 p197 j9-1 p190
Invariant 240:
c-14 p94 p124 p127 p149 p178 p179 p187 p188 j7-2 j8-1 p190
Invariant 241:
c-14 p94 p124 p127 p147 p148 p178 p179 p187 p188 j7-2 j8-1 p190
Invariant 242:
c-14 p94 p124 p178 p179 p187 p188 p197 j9-1 p190
Invariant 243:
c10 c11 c12 p61 c-14 p94 j3 p110 j11 p114 p124 p127 p149 p177 p187 j7-2 j8-1 p190

Invariant 244:
c10 c11 c12 p61 c-14 p94 j3 p110 j11 p114 p124 p127 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 245:
c10 c11 c12 p61 c-14 p94 j3 p110 j11 p114 p124 p177 p187 p197 j9-1 p190
Invariant 246:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p110 j11 p114 p124 p127 p149 p177 p187 j7-2 j8-1 p190
Invariant 247:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p110 j11 p114 p124 p127 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 248:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p110 j11 p114 p124 p177 p187 p197 j9-1 p190
Invariant 249:
c10 c11 c12 p61 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p149 p177 p187 j7-2 j8-1 p190
Invariant 250:
c10 c11 c12 p61 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 251:
c10 c11 c12 p61 c-14 p94 j3 p111 p112 j11 p114 p124 p177 p187 p197 j9-1 p190
Invariant 252:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p149 p177 p187 j7-2 j8-1 p190
Invariant 253:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p147 p148 p177 p187 j7-2 j8-1 p190
Invariant 254:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p111 p112 j11 p114 p124 p177 p187 p197 j9-1 p190
Invariant 255:
c10 c11 c12 p61 c-14 p94 j3 p110 j11 p114 p124 p127 p149 p178 p179 p187 j7-2 j8-1 p190

Invariant 256:
c10 c11 c12 p61 c-14 p94 j3 p110 j11 p114 p124 p127 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 257:
c10 c11 c12 p61 c-14 p94 j3 p110 j11 p114 p124 p178 p179 p187 p197 j9-1 p190
Invariant 258:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p110 j11 p114 p124 p127 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 259:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p110 j11 p114 p124 p127 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 260:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p110 j11 p114 p124 p178 p179 p187 p197 j9-1 p190
Invariant 261:
c10 c11 c12 p61 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 262:
c10 c11 c12 p61 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 263:
c10 c11 c12 p61 c-14 p94 j3 p111 p112 j11 p114 p124 p178 p179 p187 p197 j9-1 p190
Invariant 264:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p149 p178 p179 p187 j7-2 j8-1 p190
Invariant 265:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p111 p112 j11 p114 p124 p127 p147 p148 p178 p179 p187 j7-2 j8-1 p190
Invariant 266:
c2 c3 c4 c5 p1 p14 p31 c-14 p94 j3 p111 p112 j11 p114 p124 p178 p179 p187 p197 j9-1 p190
Invariant 267:
c-1 c-14 p94 p95 p96 j1 p98 p124 p127 p149 p188 j7-2 j8-1 p190

Invariant 268:

c-1 c-14 p94 p95 p96 j1 p98 p124 p127 p147 p148 p188 j7-2 j8-1 p190

Invariant 269:

c-1 c-14 p94 p95 p96 j1 p98 p124 p188 p197 j9-1 p190

Invariant 270:

c-1 c-14 p94 p95 p96 j1 p104 p124 p127 p149 p177 p187 j7-2 j8-1 p190

Invariant 271:

c-1 c-14 p94 p95 p96 j1 p104 p124 p127 p147 p148 p177 p187 j7-2 j8-1 p190

Invariant 272:

c-1 c-14 p94 p95 p96 j1 p104 p124 p177 p187 p197 j9-1 p190

Invariant 273:

c-1 c-14 p94 p95 p96 j1 p104 p124 p127 p149 p178 p179 p187 j7-2 j8-1 p190

Invariant 274:

c-1 c-14 p94 p95 p96 j1 p104 p124 p127 p147 p148 p178 p179 p187 j7-2 j8-1 p190

Invariant 275:

c-1 c-14 p94 p95 p96 j1 p104 p124 p178 p179 p187 p197 j9-1 p190

APPENDIX D

SIMPLE INFORMATION FLOW PATHS IN THE SYSTEM NET # 1

In this appendix the list of the simple information flow paths contained in the implemented Petri Net # 1 is presented. This list has been obtained by running a computer implementation of the algorithm described in appendix A. The simple information flow paths are obtained as an ordered list of their transitions, from a source to the sink. Then, the labelling described in chapter IV is applied to discriminate between the transitions representing functions, and those representing an artifact of the model, communication protocols, or requests for information from the database. Therefore, only the transitions representing processes are kept in each sequence of a simple informationflow path. Thus, it may happen that some reduced sequence of processes are the same. Therefore the redundant simple information flow paths are eliminated from the list. Finally, there are one hundred and two simple information flow paths left. They are listed below in the form of a source followed by a sequence of functions. Furthermore, they are grouped by type of equivalent sequence of processing.

```
c1 | f1-1-1 f10-1-1
c1 | f1-1-1 f10-1-2
c1 | f1-2-1 f10-1-1
c1 | f1-2-1 f10-1-2

c1 | f1-1-1 f11-1-1 f10-1-1
c1 | f1-1-1 f11-1-1 f10-1-2
c1 | f1-2-1 f11-1-1 f10-1-1
c1 | f1-2-1 f11-1-1 f10-1-2

c2 | f2-1-1 f2-2-3 f3-1-1 f11-1-1 f10-1-1
c2 | f2-1-1 f2-2-3 f3-1-1 f11-1-1 f10-1-2
c3 | f2-1-2 f2-2-3 f3-1-1 f11-1-1 f10-1-1
c3 | f2-1-2 f2-2-3 f3-1-1 f11-1-1 f10-1-2
c4 | f2-1-3 f2-2-3 f3-1-1 f11-1-1 f10-1-1
```

```
c4 | f2-1-3 f2-2-3 f3-1-1 f11-1-1 f10-1-2
c5 | f2-1-4 f2-2-3 f3-1-1 f11-1-1 f10-1-1
c5 | f2-1-4 f2-2-3 f3-1-1 f11-1-1 f10-1-2
c10 | f2-1-9 f3-1-1 f11-1-1 f10-1-1
c10 | f2-1-9 f3-1-1 f11-1-1 f10-1-2
c11 | f2-1-10 f3-1-1 f11-1-1 f10-1-1
c11 | f2-1-10 f3-1-1 f11-1-1 f10-1-2
c12 | f2-1-11 f3-1-1 f11-1-1 f10-1-1
c12 | f2-1-11 f3-1-1 f11-1-1 f10-1-2

c14 | f2-1-13 f9-1-3 f10-1-1
c14 | f2-1-13 f9-1-3 f10-1-2

c2 | f2-1-1 f2-2-3 f4-1-1 f9-1-3 f10-1-1
c2 | f2-1-1 f2-2-3 f4-1-2 f9-1-3 f10-1-1
c2 | f2-1-1 f2-2-3 f4-1-1 f9-1-3 f10-1-2
c2 | f2-1-1 f2-2-3 f4-1-2 f9-1-3 f10-1-2
c3 | f2-1-2 f2-2-3 f4-1-1 f9-1-3 f10-1-1
c3 | f2-1-2 f2-2-3 f4-1-2 f9-1-3 f10-1-1
c3 | f2-1-2 f2-2-3 f4-1-1 f9-1-3 f10-1-2
c3 | f2-1-2 f2-2-3 f4-1-2 f9-1-3 f10-1-2
c4 | f2-1-3 f2-2-3 f4-1-1 f9-1-3 f10-1-1
c4 | f2-1-3 f2-2-3 f4-1-2 f9-1-3 f10-1-2
c4 | f2-1-3 f2-2-3 f4-1-1 f9-1-3 f10-1-2
c4 | f2-1-3 f2-2-3 f4-1-2 f9-1-3 f10-1-1
c5 | f2-1-4 f2-2-3 f4-1-1 f9-1-3 f10-1-1
c5 | f2-1-4 f2-2-3 f4-1-1 f9-1-3 f10-1-2
c5 | f2-1-4 f2-2-3 f4-1-2 f9-1-3 f10-1-1
c5 | f2-1-4 f2-2-3 f4-1-2 f9-1-3 f10-1-2
c12 | f2-1-11 f4-1-1 f9-1-3 f10-1-1
c12 | f2-1-11 f4-1-1 f9-1-3 f10-1-2
c12 | f2-1-11 f4-1-2 f9-1-3 f10-1-1
c12 | f2-1-11 f4-1-2 f9-1-3 f10-1-2
```

```
c2 | f2-1-1 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-1
c2 | f2-1-1 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-1
c2 | f2-1-1 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-2
c2 | f2-1-1 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-2
c3 | f2-1-2 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-1
c3 | f2-1-2 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-1
c3 | f2-1-2 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-2
c3 | f2-1-2 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-2
c4 | f2-1-3 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-1
c4 | f2-1-3 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-1
c4 | f2-1-3 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-2
c4 | f2-1-3 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-2
c5 | f2-1-4 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-1
c5 | f2-1-4 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-1
c5 | f2-1-4 f2-2-3 f5-1-1 f4-1-1 f9-1-3 f10-1-2
c5 | f2-1-4 f2-2-3 f5-1-1 f4-1-2 f9-1-3 f10-1-2
c11 | f2-1-10 f5-1-1 f4-1-1 f9-1-3 f10-1-1
c11 | f2-1-10 f5-1-1 f4-1-2 f9-1-3 f10-1-1
c11 | f2-1-10 f5-1-1 f4-1-2 f9-1-3 f10-1-2
c11 | f2-1-10 f5-1-1 f4-1-1 f9-1-3 f10-1-2
c12 | f2-1-11 f5-1-1 f4-1-2 f9-1-3 f10-1-1
c12 | f2-1-11 f5-1-1 f4-1-1 f9-1-3 f10-1-1
c12 | f2-1-11 f5-1-1 f4-1-1 f9-1-3 f10-1-2
c12 | f2-1-11 f5-1-1 f4-1-2 f9-1-3 f10-1-2

c14 | f2-1-13 f7-1-2 f8-1-2 f10-1-1
c14 | f2-1-13 f7-1-2 f8-1-2 f10-1-2
c9 | f2-1-8 f2-2-1 f7-1-2 f8-1-2 f10-1-1
c13 | f2-1-12 f2-2-4 f7-1-2 f8-1-2 f10-1-1
c9 | f2-1-8 f2-2-1 f7-1-2 f8-1-2 f10-1-2
c13 | f2-1-12 f2-2-4 f7-1-2 f8-1-2 f10-1-2
c6 | f2-1-5 f2-2-1 f7-1-2 f8-1-2 f10-1-1
c7 | f2-1-6 f2-2-1 f7-1-2 f8-1-2 f10-1-1
```

```
c8 | f2-1-7 f2-2-1 f7-1-2 f8-1-2 f10-1-1
c10 | f2-1-9 f2-2-4 f7-1-2 f8-1-2 f10-1-1
c11 | f2-1-10 f2-2-4 f7-1-2 f8-1-2 f10-1-1
c12 | f2-1-11 f2-2-4 f7-1-2 f8-1-2 f10-1-1
c6 | f2-1-5 f2-2-1 f7-1-2 f8-1-2 f10-1-2
c7 | f2-1-6 f2-2-1 f7-1-2 f8-1-2 f10-1-2
c8 | f2-1-7 f2-2-1 f7-1-2 f8-1-2 f10-1-2
c10 | f2-1-9 f2-2-4 f7-1-2 f8-1-2 f10-1-2
c11 | f2-1-10 f2-2-4 f7-1-2 f8-1-2 f10-1-2
c12 | f2-1-11 f2-2-4 f7-1-2 f8-1-2 f10-1-2

c2 | f2-1-1 f2-2-3 f8-1-2 f10-1-1
c3 | f2-1-2 f2-2-3 f8-1-2 f10-1-1
c4 | f2-1-3 f2-2-3 f8-1-2 f10-1-1
c9 | f2-1-8 f2-2-1 f8-1-2 f10-1-1
c2 | f2-1-1 f2-2-3 f8-1-2 f10-1-2
c3 | f2-1-2 f2-2-3 f8-1-2 f10-1-2
c4 | f2-1-3 f2-2-3 f8-1-2 f10-1-2
c9 | f2-1-8 f2-2-1 f8-1-2 f10-1-2
c6 | f2-1-5 f2-2-1 f8-1-2 f10-1-1
c7 | f2-1-6 f2-2-1 f8-1-2 f10-1-1
c8 | f2-1-7 f2-2-1 f8-1-2 f10-1-1
c6 | f2-1-5 f2-2-1 f8-1-2 f10-1-2
c7 | f2-1-6 f2-2-1 f8-1-2 f10-1-2
c8 | f2-1-7 f2-2-1 f8-1-2 f10-1-2
c5 | f2-1-4 f2-2-3 f8-1-2 f10-1-1
c5 | f2-1-4 f2-2-3 f8-1-2 f10-1-2
```