

# **A Petri Net based Approach for Establishing Necessary Software Design and Testing Requirements\***

*S. Ramaswamy*

*Department of Computer Science, Tennessee Technological University, Cookeville, TN 38505*

*Phone: (931)-372-3691. Email: [srini@acm.org](mailto:srini@acm.org) / [srini@ieee.org](mailto:srini@ieee.org)*

**Abstract** - In this paper, a Petri net based approach for establishing the minimal design and testing requirements for software systems is presented. It is based on a technique called the minimal transition cover set (MTCS), developed from minimal transition invariants of the corresponding Petri net model [3]. The MTCS is based on identifying the important decisions being made within the design model. Depending on the design requirements, a designer may choose to expand lower level procedures during an analysis, and thereby, generate information about other "lower-level" decisions selectively within the design which are to be made available to other sub-systems. A further modification of the MTCS approach is used to derive the decision nodes that are "buried" within the design model. Again, a software test engineer may either choose to selective build test cases for individual modules and incorporate them as stubs in a higher level module, or integrate these modules within higher level modules before deriving appropriate test case scenarios. Building test cases based on the deep "decision" nodes gives the necessary test cases for system verification. The equivalence and the easier applicability of MTCS based test case generation method with the McCabe's cyclomatic complexity measure is also briefly addressed.

**Accepted to:  
Special Session on Petri Nets in Systems Design  
IEEE Conference on Systems, Man and Cybernetics  
Nashville, Oct. 2000**

---

\*: The research was conducted as part of the Virtual Environments in Education, Robotics, Automation and Manufacturing (VERAM) project, supported, in part, by grants from the National Science Foundation (#DMI-96100055) and the Center for Manufacturing Research at Tennessee Technological University.

# A Petri Net based Approach for Establishing Necessary Software Design and Testing Requirements\*

S. Ramaswamy

Department of Computer Science, Tennessee Technological University, Cookeville, TN 38505  
Phone: (931)-372-3691. Email: [srini@acm.org](mailto:srini@acm.org) / [srini@ieee.org](mailto:srini@ieee.org)

**Abstract** - In this paper, a Petri net based approach for establishing the minimal design and testing requirements for software systems is presented. It is based on a technique called the minimal transition cover set (MTCS), developed from minimal transition invariants of the corresponding Petri net model [3]. The MTCS is based on identifying the important decisions being made within the design model. Depending on the design requirements, a designer may choose to expand lower level procedures during an analysis, and thereby, generate information about other "lower-level" decisions selectively within the design which are to be made available to other sub-systems. A further modification of the MTCS approach is used to derive the decision nodes that are "buried" within the design model. Again, a software test engineer may either choose to selective build test cases for individual modules and incorporate them as stubs in a higher level module, or integrate these modules within higher level modules before deriving appropriate test case scenarios. Building test cases based on the deep "decision" nodes gives the necessary test cases for system verification. The equivalence and the easier applicability of MTCS based test case generation method with the McCabe's cyclomatic complexity measure is also briefly addressed.

## 1. Introduction

Most modern day, industrial-strength systems are complex and operate in a dynamically changing environment. Building software systems that are easier to use in such applications implies addressing the issues of increasing complexities in *managing processes and communication between processes*. In this research we are concerned with the development of a well-designed communication and coordination framework. Process management is dealt with in great detail by other research groups - such as the Capability Maturity Model, or CMM, developed by the Software Engineering Institute (SEI) at CMU. Additional references and a brief introduction to CMM are found in [4].

The objective of this paper is to develop model based software design and testing specifications for agent based systems. A good approach must also include mechanisms by which such a software entity (or unit / subsystem) can readily perceive, appreciate, understand and demonstrate cognizant behavior while working with other system entities. Such an entity is termed a "software agent" in this paper. In spite of the various definitions and interpretations of this term, we must be able to define some of the innate attributes that characterize an agent in *any* system. Thus, instead of trying to define what a Software Agent (SA) is, a software agent is defined to be any system component (or part, or subsystem, etc.) that appropriately answers the question "aRe yoU

A Software Agent? (R U A SA?)" where, R represents the ability to Represent, U the ability to Understand, and A the ability to Act. In detail, these attributes mean the following: (i) **Represent**: Represent both behavioral and structural characteristics of the agents comprising the system. (ii) **Understand**: Understand both local and system objectives, prioritize them according to available knowledge about the other system agents, the environment and itself. (iii) **Act**: Perform either deliberative or reactive actions based on the agent's knowledge, with respect to events generated by itself, the environment and other system agents. Thus an agent, *in our perspective*, is akin to an object (class / group) with appropriate mechanisms for exhibiting intelligent behavior. Therefore, objects can be used to implement agents.

Several modeling tools and methodologies, based on object-oriented techniques, have been developed for system specification and analysis [13-14]. Complete object-oriented methodologies (from analysis through design) are discussed in [15-41]. A responsibility-based approach is presented in [38, 41] to define objects during systems development. A brief taxonomy of the characteristics to be considered when grouping concepts as objects is also presented in [38]. In [22], an iterative method for object-oriented analysis and design is presented, which emphasizes proper construction, generalization and abstraction of classes. Current agent-based design methodologies extend the object-oriented design approach to intelligent agents operating in distributed environments [28-29]. For a detailed introduction to software agents, their definitions and classifications, etc. the interested reader is referred to [28,39,40] and references therein.

In this paper, we are interested in developing mechanisms that help in defining the agent requirements and establishing necessary software design and testing requirements based on their respective Petri net models. The approach presented in this paper derives heavily from Petri net theory and the design of systems using PN invariants. For an in-depth treatment of Petri net invariants and related literature the reader is referred to [1] and references therein. The Petri net-based approach offers the following advantages: (i) Agents (classes / objects) in the software system can be defined, in structure and behavior, in close correspondence to real-world entities. (ii) Model and sub-model reuse by means of analogy, decomposition, or synthesis, will aid in the rapid development of new software elements. (iii) Object inheritance during the implementation will enable the creation of new objects and relationships with minimal effort (iv) Data and process encapsulation will provide the appropriate distinction between object boundaries and will be effective in identifying, monitoring and controlling the propagation of errors. (v) Finally, the approach will help establish well-defined communication mechanisms and to set up data and process security features.

---

\*: The research is part of the Virtual Environments in Education, Robotics, Automation and Manufacturing (VERAM) project, supported, in part, by grants from the National Science Foundation (#DMI-96100055) and the Center for Manufacturing Research at Tennessee Technological University.

<p><b>Selection:</b></p> <p><b>Comments:</b> Derivation of choice sets</p> <p><b>Input:</b> All T-invariants of the agent model</p> <p><b>Output:</b> A choice set whose entries are sets of transitions that are possible candidates for the minimal cover set.</p> <p><b>Algorithm:</b></p> <p><b>Step 0:</b> Construct a table where the rows correspond to the different T-invariants(<math>x_i</math>) of the system (<math>1 \leq i \leq \#T</math>-invariants) and the columns correspond to the distinct transitions(<math>t_j</math>) in the PN system model (<math>1 \leq j \leq \#transitions</math>). Table entries are marked (with entry 1) when a particular transition appears in the corresponding T-invariant. Let the choice set, <math>\psi</math>, initially be a null set.</p> <p><b>Step 1:</b> Identify all columns that have the same column entries. Mark the corresponding transitions as equivalent transitions and eliminate all but one of those columns from the table.</p> <p><b>Step 2:</b> Select all columns that have the maximum number of column entries (maximum number of 1's) that correspond to distinct <math>x_i</math>'s. Include all the transitions in <math>\psi</math>. For every transition in <math>\psi</math>, do step 3.</p> <p><b>Step 3:</b> Select all columns with minimum number of column entries (minimum number of 1's). For every such transition, if any of the corresponding <math>x_i</math> entries are not covered by any of the transitions in <math>\psi</math>, include the transition in <math>\psi</math>.</p> <p><b>Step 4:</b> Delete all columns identified in step 3 and update the table.</p> <p><b>Step 5:</b> Repeat steps 3 and 4 until all <math>x_i</math>'s are covered.</p> <p><b>Step 6:</b> For every transition <math>t_i</math> in <math>\psi</math>, replace <math>t_i</math> by the set <math>t_i, t_j, t_k, \dots</math> etc.; where <math>t_j, t_k</math> are equivalent transitions identified in step 1. This gives the final choice set <math>\psi</math>.</p>	<p><b>Pruning:</b></p> <p><b>Comments:</b> Let <math>S = S_1, S_2, S_3, \dots</math> be the set of ordered sets derived from <math>\psi</math>. The elements of <math>S_1, S_2,</math> and <math>S_3</math> are distinct non-equivalent entries in <math>\psi</math>. The sets <math>S_1, S_2, S_3, \dots</math>, correspond to all possible combinations of the elements from the choice set, <math>\psi</math>. Let <math> S  = i</math>. The MTCS is one of the subsets of <math>S</math>.</p> <p>Let <math>E</math> be the set of events in the system. All events <math>e_i \in E</math> are associated with some transition in the system model. If a transition is not associated with the occurrence of an event <math>e_i \in E</math>, then it is assumed to be fired by a universal event, <math>\xi</math>.</p> <p><b>Input:</b> Choice set derived by the selection algorithm.</p> <p><b>Output:</b> Minimal Transition Cover Set (MTCS):</p> <p><b>Algorithm:</b></p> <p><b>Step 0:</b> Eliminate all sets in <math>S</math> that include a temporary transition (temporary transitions are often added to a PN system model to build the reachability graph and studying PN properties such as reversibility). If <math> S  = 1</math>, return the subset of <math>S</math> as the MTCS, else go to step 1.</p> <p><b>Step 1:</b> Check to see if <i>all</i> remaining sets in <math>S</math> contain at least a transition that is fired only by <math>\xi</math>. If so, continue to step 2. Otherwise, (there are sets in <math>S</math> that do not contain any transitions fired only by <math>\xi</math>) eliminate all the sets in <math>S</math> that contain at least one transition fired only by <math>\xi</math>. If <math> S  = 1</math> return the subset of <math>S</math> as the MTCS.</p> <p><b>Step 2:</b> Calculate the total invariant cover (TIC) for every subset in <math>S</math>. The TIC is derived as follows: For every distinct pair of transitions contained in a subset of <math>S</math>, calculate the common cover, which is the number of common column entries for corresponding <math>x_i</math>'s. TIC is the sum of all such common covers for a given subset of <math>S</math>. Let <math>m</math> be the least TIC for the subsets in <math>S</math>. Eliminate all subsets in <math>S</math> that have a TIC <math>&gt; m</math>. If <math> S  = 1</math> then return the subset of <math>S</math> as the MTCS; else return any one subset of <math>S</math> as the MTCS.</p>
<b>The Selection Algorithm</b>	<b>The Pruning Algorithm</b>

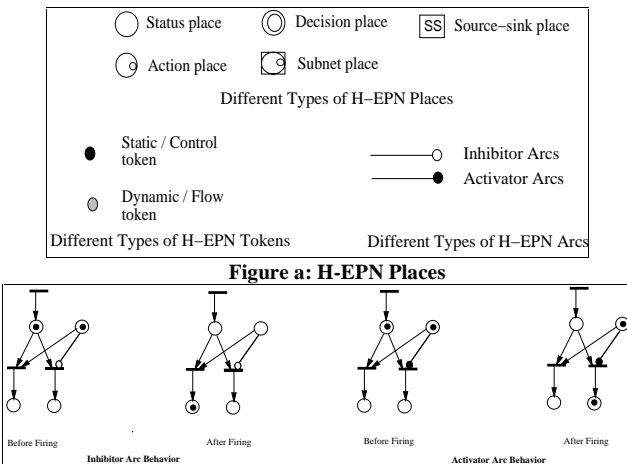
**Figure 1. The MTCS Algorithm**

The organization of this paper is as follows: Section 2 briefly presents the minimal transition cover set (MTCS) algorithm, previously presented in [3]. Modifications and extensions to the

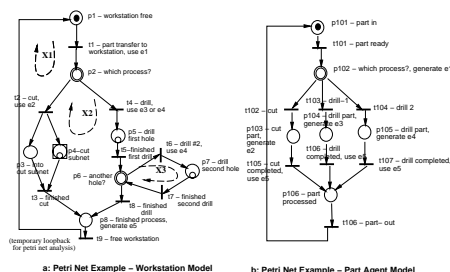
MTCS algorithm are used to derive the agent design and testing requirements; presented in this paper. Section 3 presents the application of the MTCS algorithm to developing software design and testing requirements. It presents the application of the approach to both structured and OO design paradigms. Section 4 concludes the paper.

## 2. The Minimal Transition Cover Set

Petri Nets (PNs) are powerful behavior description formalisms for systems modeling and analysis. In this paper we use



**Figure b: H-EPN Arc Behaviors**  
**Figure 2. H-EPN Extensions**



**Figure 3. MTCS Example**

Hierarchical Time-Extended Petri Nets (H-EPNs), a form of extended PN. H-EPNs have been chosen for the following reasons: *i)* H-EPNs may be easily transformed to classical PN models, and therefore, traditional PN analysis techniques are applicable to H-EPNs. *ii)* H-EPNs provide a structured approach to combining PN models developed through top-down and bottom-up design techniques. The interested reader is referred to [2, 3] for details on H-EPNs and a detailed introduction to MTCS.

The minimal transition cover set (MTCS) algorithm gives the minimum number of transitions that cover all the distinct (independent) subflows of a given system model. Figure 1 presents the MTCS algorithm. The calculation of the MTCS is a two step process: *(i) Selection:* The selection algorithm is used for the selection of the choice sets that contain transition groups that account for all the different independent subparts of the system model. and *(ii) Pruning:* The pruning algorithm is used in determining a subset of transitions from the choice set that minimally cover all the independent subparts of the system. The pruning algorithm may be additionally augmented with *domain dependent heuristics* (depending on what information is critical) to find the best minimal cover. The output of the pruning algorithm is the MTCS. Figure 3 presents a small example and illustrates the steps involved in deriving the MTCS. The use of the MTCS approach in the identification of critical system areas for monitoring and control has already been presented in [3].

### 3. Software Design and Testing Issues

For two software agents to demonstrate domain specific intelligence and to act (react) effectively and efficiently to their environment, a system agent needs to have some knowledge about processes in other agents. In reality however, each agent cannot get and does not need information about *every* process in other system agents. While some processes in these other agents will have an impact on the decisions being made at the agent in question, not all processes may be relevant. Therefore, there needs to be a way by which one must be able to determine information about these critical processes that affect decision making in other system agents. Given this need, it might be very apt, if an agent can at least

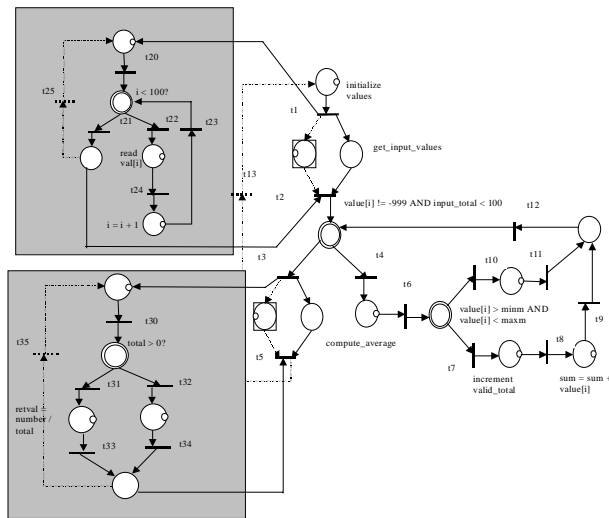


Figure 6. H-EPN Model of find\_average Procedure

```

procedure find_average
value [100], i, input_total = valid_total = 0, sum = 0;
get_input_values(value);
DO WHILE ( (value[i] != -999) AND (input_total < 100) {
  increment input_total;
  IF ( (value[i] >= minm) AND (value[i] <= MAXM)) {
    increment valid_total;
    sum = sum + value[i];
  }
  else skip;
ENDIF
increment i;
ENDDO
average = compute_average(sum, valid_total);
END find_average

procedure get_input_values
return type: void
input parameters: val []
local parameters: i
FOR (i = 0; i < 100; i++) read
val[i];
return;
END get_input_values

procedure compute_average
return type: double
input parameters: number, total
local parameters: retval;
IF ( total > 0)
  retval = number/total;
else retval = -999;
return retval;

```

Figure 4. Example: The find\_average Procedure

1. input value[k] = valid input, where k < j, and value [j] = -999 where 2 ≤ j ≤ 100.
2. value[1] = -999.
3. attempt to process more than 101 values.
4. try to input more than 100 values.
5. value[k] < minm.
6. value[k] > maxm.
7. input "n" values to see how average is computed.

Figure 5. Test Criterion for the find\_average Example

get_input_val ues	compute_aver age	find_average	Combined Model
X1: t20, t21, t25	X3: t30, t31, t33, t35	X5: t1, t2, t3, t5, t13	X1: t1, t2, t3, t5, t13, t20, t21, t30, t31, t33
X2: t22, t23, t24	X4: t30, t32, t34, t35	X6: t4, t6, t7, t8, t9, t12	X2: t1, t2, t3, t5, t13, t20, t21, t30, t32, t34
		X7: t4, t6, t10, t11, t12	X3: t4, t6, t10, t11, t12
			X4: t4, t6, t7, t8, t9, t12
			X5: t22, t23, t24

Table 1. T-Invariants for the find\_average Example

know if any events relating to these critical processes are being executed at the other agent. To apply the MTCS algorithm to software design and testing different strategies (scenarios) are derived. Common to all the three strategies is the need to derive the transition invariants of the agent. Once the transition invariants are derived the different strategies may be used depending upon the needs of the application at hand.

#### 3.A. MTCS in the Design and Testing of Simple Procedures

To illustrate the approaches more thoroughly, the following procedural example (adapted and modified from [4]) is used. The program is to compute the average of those numbers in an array of 100 numbers, which lie within a given range (min, max). A array value of -999 indicates the end of the values in the array.

The widely used McCabe's cyclomatic complexity measure presented in [5], for the above procedure is 7. The cyclomatic complexity measure gives the number of linearly independent paths in the program flow graph. It is a very useful measure for building test cases for programs such that each such linear path is exercised at least once. Using McCabe's measure, the above procedure will need 7 test cases to test the program. These test criteria are enumerated in Figure 5. Out of these 7 test criteria, test cases 1 and 2 can be designed as part of the other test criteria, and hence if test cases were built for test criterion 3-7, it can be ensured that all the linearly independent paths in the program are traversed.

The H-EPN models for the compute\_average, get\_input\_values and find\_average procedures are illustrated Figure 6. Table 1 gives the minimal T-invariants for the procedures in above example. The T-invariants for procedures have been derived independently and in combination with the find\_average procedure. These transition in the minimal T-invariant are stored as a linked list of elements where each transition has a counter to indicate the number of minimal invariants it covers.

Minimal T-Invariants	
X1: t2, t34; t1, t32	X6: t28, t30; t19, t25, t26
X2: t4, t35; t3, t31, t33	X7: t29, t30; t19, t25, t27
X3: t6; t5	X8: t10, t12; t7, t8
X4: t24; t20, t22	X9: t11, t12; t7, t9
X5: t23, t24; t20, t21	X10: t16, t18; t13, t14
	X11: t17, 18; t13, t15

**Table 3. Minimal T-Invariants for Class Monitor – I**

### 3.A.1 Greedy Choice Strategy:

The greedy choice strategy, as the name suggests, follows a greedy approach to choosing transitions that will belong to the MTCS. That is, the first transition that maximally covers the T-invariants is chosen to be included in the MTCS. The search ends when transitions that cover all the minimal T-invariants for the system agent model are included in the MTCS. While this approach may do well on most occasions when the PN model is constructed carefully. However, this algorithm vastly derives on the spatial ordering of the transitions within each T-invariant.

Using the greedy choice strategy on the T-invariants in Table 1, we get t20, t22, t30, t1 and t4 as the MTCS transitions while looking at the procedures independently, and, t1, t4, and t22 while looking at the combined model. However, this differentiation does not make much of a change in the knowledge that can be derived from the model, since the minimal T-invariants X1 and X2 for the combined model are a combination of invariants X1, X3, X4 and X5 in the independent procedures. In either case, while t4 and t22 (representing the decision loops in the model) do provide some information which might be relevant for other procedures about the operational status of the averaging process, the other transitions in the MTCS (derived using this strategy) do not convey much information.

If events are associated with the transitions in the model then a condition to include only those transitions that are event enabled can be imposed. However, even this may not provide all the necessary information for design / testing of the software developed. Also, the choice for the MTCS in this case will be affected by randomly chosen labels (names) for the transitions, the ordering mechanism followed by the minimal T-invariant generation algorithm, etc. Given the above reasons, this approach will not lead to a very accurate set of agent processes as required for the design / testing of these agents. For this reason, the design and testing strategies are developed. These techniques define additional heuristics for choosing transitions that belong to the MTCS that make it very useful in the design and testing requirements using PN models of agent based systems.

### 3.A.2 Design Choice Strategy:

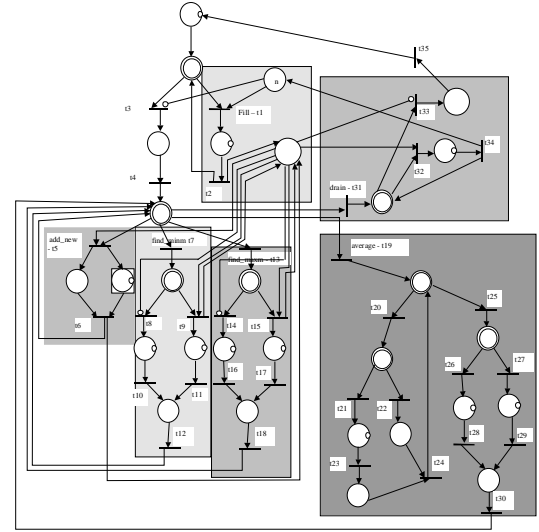
In the design choice and testing choice strategies, the transitions in the minimal T-invariants of the agent model are split into two subsets. The first subset consists of transitions that are not output to any decision places in the H-EPN model. The second subset consists of transitions that are output to decision places in the

get_input_values	compute_average	find_average	Combined Model
X1: t20, t25; <b>t21</b>	X3: t30, t33, t35; <b>t31</b>	X5: t1, t2, t5, t13; <b>t3</b>	X1: t1, t2, t5, t13, t20, t30, t33; <b>t3</b> , t21, t31
X2: t23, t24; <b>t22</b>	X4: t30, t34, t35; <b>t32</b>	X6: t6, t8, t9, t12; <b>t4</b> , t7	X2: t1, t2, t5, t13, t20, t30, t34; <b>t3</b> , t21, t32
		X7: t6, t11, t12; <b>t4</b> , t10	X3: t6, t8, t9, t12; <b>t4</b> , t7
			X4: t6, t11, t12; <b>t4</b> , t10
			X5: t23, t24; <b>t22</b>

**Table 2. Ordered minimal T-Invariants for the find\_average Procedure**

1. Generates the MTCS using the design choice strategy.
2. Mark MTCS transitions that cover more than one minimal invariant.
3. For all transitions that cover more than one T-invariant repeat step 4 until the transitions chosen cover exactly one minimal T- invariant.
4. For the minimal T-invariants covered by this transition in the MTCS, choose a set of transitions (which are an output to a decision place) that exactly cover only one T-invariant. (Note: Scan down the list of transitions repeatedly choosing the next transition until the counter for the transition is 1).

**Figure 7. Enhanced MTCS Algorithm for the Testing Choice Strategy**



**Figure 8. PN Model of Class Monitor - I**

model. This is illustrated in Table 2.

In choosing transitions that belong to the MTCS for the design choice strategy, preference is given to decision places that cover the maximum number of T-invariants. This selection strategy gives all the critical high level decisions that are made in the agent model. The knowledge about the agent that needs to be available for other agents is derived from the decision processes input to these transitions. Depending upon how the PN model is derived, other system agents either need to know about the decision place that represents the process itself or the outcome of the decision process (the process which is begun by the chosen MTCS transition).

```

class monitor {
private:
    temp_value[100];
public:
    monitor();
    ~monitor();
    find_minm();
    find_maxm();
    add_new();
    find_average();
}

```

**Figure 9. OO Design – I**

Using the design choice strategy on the T-invariants Table 1 we get t21, t22, t31, t32, t1 and t4 as the MTCS transitions. In the combined model, we derive t3, t4 and t22 as the MTCS choices. Depending upon the information that is required about each agent and the agent design restrictions, the designer could choose to expand procedures or use them independently. The maximal information about an agent is derived when all the procedures within an agent design are treated independently while the minimally necessary information is derived when the procedures are expanded and the combined model is used to derive the MTCS.

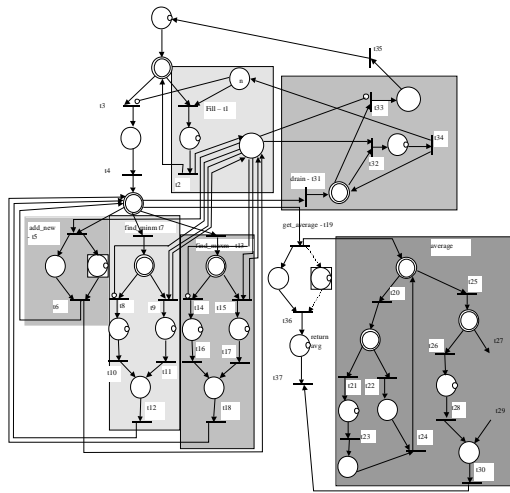


Figure 10. PN Model of Class Monitor - II

### 3.A.3 Testing Choice Strategy:

This strategy also derives from the ordered minimal invariants generated in Table 2. However, unlike the design choice strategy, which included transitions which are input to the decision places that maximally cover all invariants, the choice for transitions which belong to the MTCS in this strategy is different. Here, only those transitions that are input from decision places, which minimally cover each transition invariant, are chosen to be in the MTCS. This enhancement of the MTCS algorithm is illustrated in Figure 7. Due to the nature of the minimal T-invariants, it is guaranteed that there will be at least one transition with a counter value of 1. This is because, if there were no such transition with a counter value of 1, then the minimal T-invariant chosen will not be unique, since each minimal T-invariant provides exactly one linearly independent path

Greedy Strategy	Design Choice Strategy*	Testing Choice Strategy
t1, t3, t5, t20, t19, t7, t13	t1, t3, t5, t20, t19, t7, t13	t1, t3, t5, t21, t22, t26, t27, t8, t9, t14, t15

\*: Produces the same output as the greedy choice strategy due to the labeling of the transitions and their ordering by the minimal transition invariant generation algorithm.

Table 4. MTCS Transitions – I

in the PN model.

Using this strategy the MTCS transitions are t21, t22, t31, t32, t3, t7, and t10. In the combined model the MTCS transitions are t31, t32, t7, t10 and t22. These correspond to the innermost conditions being checked in the decision logic. Building test cases for decisions associated with these conditions will guarantee the fact that all the linearly independent paths of the logic are tested at least once. Testing the decisions associated with the MTCS transitions chosen with this strategy correspond to the test criterion developed using the McCabe’s cyclomatic complexity measure as shown in Figure 6. Test cases exercising transitions t31 and t32 check for a valid total, test cases for t22 checks for a valid number of inputs,

```

class monitor {
private:
    temp_value[100];
    avg;
public:
    monitor();
    ~monitor();
    find_min();
    find_max();
    add_new();
    get_average();
private:
    find_average();

```

and test cases for t7 and t10 check for values within a given range. Thus building test cases based on this MTCS strategy allows the establishment of necessary test conditions for software design.

Figure 11. OO Design – II

Minimal T-Invariants for Class Monitor		T-Invariants for Procedure Average
X1: t2, t34; t1, t32	X5: t10, t12; t7, t8	X9: t24; t20, t22
X2: t4, t35; t3, t31, t33	X6: t11, t12; t7, t9	X10: t23, t24; t20, t21
X3: t6; t5	X7: t16, t18; t13, t14	X11: t28, t30; t25, t26
X4: t36, t37; t19	X8: t17, t18; t13, t15	X12: t29, t30; t25, t27

Table 5. Minimal T-Invariants for Class Monitor – II

Greedy Strategy	Design Choice Strategy*	Testing Choice Strategy
t1, t3, t5, t19, t7, t13	t1, t3, t5, t19, t7, t13	t1, t3, t5, t8, t9, t14, t15

\*: Produces the same output as the greedy choice strategy due to the labeling of the transitions and their ordering by the minimal transition invariant generation algorithm.

Table 6. MTCS Transitions – II

### 3.B. MTCS Application to OO Designs: An Example

In this section, the MTCS approach is used in studying OO design issues. The approach allows designers to distinguish between class services that need to be public versus those that need to be declared to be private and qualitatively evaluate two design models. To illustrate, we choose an automated power plant operation. There is a need to periodically record (sample) temperature and pressure values in the boilers and use this for tuning their temperature and pressure readings appropriately. The class responsible for this monitoring operation keeps a circular list of the most recent values (ex. 100). Whenever a trend that deviates from a constant requirement is detected, the object triggers some remedial action.

A first cut design for the monitor class might include an encapsulated array (or list) for storing the values, a procedure to add a new value sampled, and procedures to find the maximum, minimum and average of the values recorded. Figure 8 provides the PN model for the interactions between the various procedures. The ordered minimal T-invariants for the model is presented in Table 3 and the MTCS transitions based on this minimal invariant cover is presented in Table 4. Based on the design choice strategy derived in Table 4, the design of the monitor class design would be as shown in Figure 9.

Figure 10 provides a modified version of the class model presented in Figure 8. Table 6 Table 7 provides the minimal T-invariants and the MTCS transitions for the model. A Class design based on this model is shown Figure 11

## 4. Conclusions

This paper presents the appropriateness of the MTCS approach to designing systems that can exhibit selective information sharing between their entities (or agents). Agents, by virtue of their definition, need to interact with other such agents in any system. Therefore, it is imperative that an agent puts forth some meaningful information about itself to other system agents whose decisions are affected by its activities (or processes). Since hierarchical PNs can model system details at any level of granularity they present an ideal tool for designing and testing agents at any abstraction level. The MTCS transitions can be used as a basis for designing and testing these agent based systems. While the design choice strategy

Greedy Strategy	Design Choice Strategy*	Testing Choice Strategy
t20, t25	t20, t25	t21, t22, t26, t27

\*: Produces the same output as the greedy choice strategy due to the labeling of the transitions and their ordering by the minimal transition invariant generation algorithm.

Table 7. MTCS Transitions for Procedure Average – II

may help determine those pieces of information that need to be broadcast about any given agent, the testing choice strategy may be used to design the necessary test cases for testing each agent's functionality.

The approach can also be used to compare and contrast two agent designs with respect to issues such as information hiding, encapsulation, member function access privileges and restrictions etc. While the first cut design in Figure 9 might have been the simplest and easiest to implement, the design shown in Figure 11 is much better for a good implementation since this design restricts access to the function that computes the average.

## References

1. T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol. 77, No. 4, April 1989, pp. 541-580.
2. S. Ramaswamy, "Hierarchical Time-Extended Petri Nets for Integrated Control and Diagnostics of Multi-Level Systems", Ph.D. Thesis, The Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504, May 1994.
3. S. Ramaswamy, K. S. Barber, "An Approach for Monitoring and Control of Agent Based Systems", *Proceedings of the 1997 Robotics and Automation Conference, Albuquerque, NM, April 1997*.
4. R. S. Pressman, "Software Engineering: A Practitioner's Approach", Third Edition, August 1996, McGraw Hill.
5. T. McCabe, "A Complexity Measure", *IEEE Trans. on Software Engineering*, Vol. 2, 1996.
6. Braham, R., and Comerford, R., "Sharing Virtual Worlds", *IEEE Spectrum*, pp.18-25, March 1997.
7. Greshon, N., Eick, S., "Visualization's New Track: Making Sense of Information", *IEEE Spectrum*, Nov 1995, pp. 38-56.
8. Marhefka, D., W., and Orin, D. E., "XAnimate: An Educational Tool for Robot Graphical Simulation", *IEEE Robotics & Automation Magazine*, pp.6-14, Vol.3, No.2, June 1996.
9. ---, "Semiotic Modeling of Sensible Agents", *Intelligent Systems: A Semiotic Perspective*, NIST, Gaithersburg, MD, Oct. 1996. (Organizers: Dr. K. S. Barber and Dr. S. Ramaswamy)
10. Ahn, J.S., Ramaswamy, S., Crawford, R. H., "A Formalism for Modeling Engineering Design Processes", *Accepted to the IEEE Transactions on Systems, Man and Cybernetics (SMC)*, July 1996.
11. Andersson, C. Carlsson, O. Hagsand and Olov Stahl, "The Distributed Interactive Virtual Environment Technical Reference Manual", Swedish Institute of Computer Science, March 1994.
12. Barcio, B. T., Ramaswamy, S., Barber, K. S., "An Object-Oriented Modeling and Simulation Environment for Reactive Systems Development", *Intl. J. of Flexible Manuf. Systems*, Vol. 9, No. 1, Jan. 1997, pp. 51-80.
13. Barcio, B. T., Ramaswamy, S., Macfadzean, R., Barber, K. S., "Object-Oriented Modeling and Simulation of a Notional Air Defense System", *SIMULATION*, Vol. 66, No 1, Jan 1996, pp. 5-21.
14. Bear, S., Allen, P., Coleman, D., Hayes, F., "Graphical Specification of Object Oriented Systems", *ECOOP/OOPSLA 1990 Proceedings, Oct. 1990*, pp. 28 - 37.
15. Booch, G., "Object Oriented Design with Applications", Redwood City, California: The Benjamin/Cummings Pub. Co., Inc., 1991.
16. Brave, Y., Heymann, M., "Control of Discrete Event Systems Modeled as Hierarchical State Machines", *IEEE Transactions on Automatic Control*, vol. 38, no. 12, Dec. 1993, pp. 1803 - 1819.
17. Cherry, G. W., "Stimulus-Response Machines: A New Visual Formalism for Describing Classes and Objects", *ACM SIGSOFT Software Engineering Notes*, vol. 18, no. 2, Apr. 1993, pp. 86 - 95.
18. Coleman, D., Hayes, F., Bear, S., "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design", *IEEE Transactions on Software Engineering*, vol. 18, no. 1, Jan. 1992, pp. 9 - 18.
19. Doyle, J., "Rationality and its Roles in Reasoning", *Computational Intelligence*, Vol. 8, No. 2, pp. 376-409, 1992.
20. Fang, Y., and Liou, F.W., "Virtual Prototyping of Mechanical Assemblies with Deformable Components", *Journal of Manufacturing Systems*, pp.211-219, Vol.16, No.3, 1996.
21. Goldstein, N., Alger, J., "Developing Object-Oriented Software for the Macintosh: Analysis, Design, and Programming", Reading Massachusetts: Addison-Wesley Publishing Company, Inc., 1992.
22. Gossain, S., Anderson, B., "An Iterative-Design Model for Reusable Object-Oriented Software", *ECOOP/OOPSLA 1990 Proceedings, Ottawa, Canada, Oct. 1990*, pp. 13 - 27.
23. Harel, D., Pnueli, A., Schmidt, J. P., and Sherman, R., "On the Formal Semantics of Statecharts", *Proc. 2nd IEEE Symp. on Logic of Computer Sci.*, 1987, pp. 54 - 64.
24. Harel, D., Pnueli, A., "On the Development of Reactive Systems", *Logics and Models of Concurrent Systems, NATO, ASI Series, vol. 13, K.R. Apt, Ed. Springer-Verlag, New York, 1985*, pp. 477 - 498.
25. Harel, D., "On Visual Formalisms," *Comm. of the ACM*, Vol. 31, no. 5, May 1988, pp. 514 - 531.
26. Harel, D., "Statecharts: A Visual Formalism for Complex Systems," *Sci. Comput. Program.*, vol. 8, no. 3, June 1987, pp. 231 - 274.
27. Harel, D., et al., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", in *Software State-of-the-Art: Selected Papers*, DeMarco, T., and Lister, T., editors, Dorset House, 1990, pp. 322-338.
28. Huhns, M. N., Singh, M. P., "Agents on the Web: Agents are Everywhere!" <http://www.computer.org/internet/9701/agents9701.htm>
29. Jernigan, S. R., Ramaswamy, S., Barber, K. S., "A Distributed Search and Simulation Method for Job Flow Scheduling", *SIMULATION*, June 1997.
30. Karr, R., Reece, D., and Franceschini, R., "Synthetic Soldiers", *IEEE Spectrum*, pp.39-45, March 1997.
31. Kraus, S., and Wilkenfeld, J., "Negotiation over Time in a Multi Agent Environment", Technical Report UMIACS-TR-91-51, University of Maryland, Institute of Advanced Computer Studies, 1991.
32. Ormes, J., Ramaswamy, S., Cheng, T., "Modeling and Simulation of Manufacturing Systems in Interactive Virtual Environments", *6th IEEE International Conf. on Emerging Technologies and Factory Automation, UCLA, Los Angeles, September 1997*.
33. Ramaswamy, S., Valavanis, K. P., "Modeling, Analysis and Simulation of Failures in a Materials Handling System with Extended Petri Nets", *IEEE Trans. on Systems, Man, and Cybernetics*, pp.1358-1373, Vol.24, No.9, September 1994.
34. Robertson, G. G., et. al. "Information Visualization Using 3-D Interactive Animation", *Comm. of the ACM*, Vol. 36, No. 4, April 1993, pp. 57-71.
35. Rosenschein, J. "Rational Interaction: Cooperation Among Intelligent Agents", Ph. D. Thesis, TR#: STA-CS-85-1081, Stanford Univ., 1986.
36. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., "Object-Oriented Modeling and Design", Englewood Cliffs, New Jersey: Prentice Hall, 1991.
37. Sen, S., Durfee, E., "A Formal Study of Distributed Meeting Scheduling", *Group Decision and Negotiation Support Systems*, 1996.
38. Sharble, R. C., Cohen, S. S., "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", *ACM SIGSOFT Software Engg. Notes*, vol. 18, no. 2, Apr. 1993, pp. 60 - 73.
39. Shoham, Y. "Agent-Oriented Programming", *Artificial Intelligence*, Vol. 60, pp. 51-92, 1993.
40. Sycara, K., and Zeng, D., "Coordination of Multiple Intelligent Software Agents", *Internl. Jour. Cooperative Information Systems*, Vol. 5, Nos. 2/3, 1996, pp. 181-212.
41. Wirfs-Brock, R., Wilkerson, B., Weiner, L., "Designing Object-Oriented Software", Prentice-Hall, 1990.