

Software Design and Testing Using Petri Nets: A Case Study Using a Distributed Simulation Software System *

S. Ramaswamy**, R. Neelakantan

Software Automation and Intelligence Laboratory, Department of Computer Science
Tennessee Technological University, Cookeville, TN 38505

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

ABSTRACT: - This paper presents the application of Petri Nets in software design and testing using a distributed simulation application for naval radar interference resolution as a case study. Good software systems design promotes the “minimal” exertion of control over the different subsystem components; nevertheless, the knowledge of “critical” states of operation within one subsystem can orient other subsystems to react intelligently. Hence, a methodology for determining all the important “logical areas” of a subsystem is most crucial. The approach used here is based on a technique called Minimal Transition Cover Sets, or MTCS, which is a refinement of the minimal transition invariants of the corresponding Petri Net model of the system [1]. It allows for selective information sharing; one of the most important attributes of good object-oriented design. This paper explores this approach to aid the system designer build some high level knowledge about the different subsystem components. Using this high-level knowledge, the designer can then choose an appropriate design from a group of designs and thus can be in a position to qualitatively compare and contrast two or more design in terms of information sharing capabilities [2]. A further enhancement of this MTCS approach involves its use in building software test cases. It allows for the development of the *necessary* test cases in the verification of software system modules.

I. INTRODUCTION

Software systems designed to be distributed, interactive and intelligent (in a domain-specific sense), with ubiquitous human interfaces and the ability to exhibit intelligent cooperative behaviors will pave the way for achieving agile and flexible automation. With the advent of complex systems in modern day industry that operate in dynamically changing environments, software development for such systems calls for addressing the issues of increasing complexities in managing processes and communication between processes. It is becoming all the more imperative that the software systems be robust and error free in such real-world applications. Thus,

demonstrating intelligent decision making capabilities while working with other subsystem entities is critical. This research is part of an effort towards the development of such flexible software systems. Thus, developing methods, tools and methodologies that aid in the design of software systems using such scalable, basic underlying structures is the principle objective of this research.

Such software modules (or objects, procedures, subsystems as referred to in different programming and development paradigms) that make up a flexible software system are termed Intelligent Coordinating Entities, or ICE. The ICE concept is based on the observation that if software modules can be designed with a simple, well-defined underlying symmetrical coordination framework, (similar to the formation of snow crystals that come in infinitely different shapes and combine in various ways to form snowflakes) then they can be combined in various ways to implement different kinds of software applications in a multitude of application domains.

This paper¹ is organized as follows: Section II provides an overview of the Intelligent Coordinating Entities (ICE) concept, the use of Petri Nets for implementing ICE elements and their applicability to designing better software systems. Section III provides a brief introduction to the example distributed simulation system, the overall Petri Nets model of the system and its various subnets. Section IV provides the design analysis as well as test case design for “necessary” software verification. Section V concludes the paper.

II. INTELLIGENT COORDINATING ENTITIES

The ICE design concept goes a step further than current agent based, and other related, paradigms to designing intelligent systems. ICE-based design seeks to extend current agent-oriented design approaches to implicitly include a **structured** coordination framework, which can be implemented through different enabling technologies (such as DCOM/CORBA/JINI) - necessary to bring such intelligent coordinated behaviors among constituent modules (ICE elements, or ICICLES) in modern-day systems to fruition. In a system, an ICICLE, therefore, should not only consist of a representation,

* The research is supported, in part, by the Center for Manufacturing Research at Tennessee Technological University.

** Please address all correspondences to Dr. S. Ramaswamy.

¹ Due to restrictions in length, throughout this paper, it is assumed that the reader is familiar with at least the basic concepts of Petri Nets. The references at the end provide abundant additional introductory material to Petri Nets [3, 4].

decision and reaction structure, but also include the mechanisms to orchestrate such reactions in a coordinated fashion using a well-defined communication structure as the underlying basis. Thus, a communication-centric structure forms the core of every ICE element (similar to the hexagonal structure in the snow crystal analogy). Depending on the enabling technology used, it may or may not be necessary to include the code for the communication structure within each ICICLE. However, depending upon the other players in the coordination, an ICICLE can selectively choose the appropriate information it is willing to share with the group by means of a dynamic "information-sharing" model. There are two fundamental issues to designing such ICE elements - its communication and coordination structure.

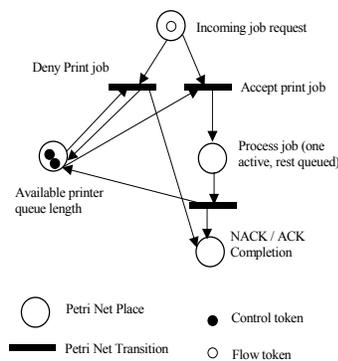


Figure 1. Petri Net Model of a Simple Print Process Functionality

The basis for every ICE element is a well-defined, flexible communication structure. Therefore, a software system designed with ICE elements, or ICICLES, assumes a networked structure of dynamically interacting ICICLES. The interaction capability of an ICICLE may be either time-invariant or time-dependent.

Hence, it may be different at different time instances depending upon the composition and needs of the system (may be based on system goals and/or objectives) and the other elements involved in the interaction. Each ICICLE may or may not have a corresponding hardware resource associated with it - the implicit assumption being that all hardware devices have an inherent ability to communicate with other active elements within the group. This communication capability may be platform and language-independent and hence can be brought forth by using any standard specifications such as DCOM, CORBA, JINI, etc., that provide immense distributed coordination capabilities.

The communication framework, while providing the flexibility of interaction, does not by itself *make* an ICICLE intelligent. Thus, on top of the above mentioned communication framework, a well-defined coordination capability has to be built. This is to be implemented by maintaining a dynamic "information-sharing" model accessible by each ICE element. This model is to be built and/or updated periodically and possibly maintained consistently across a network. Any ICICLE will use this

information to verify the availability of a service or other such information. Servers may provide the necessary capabilities for model maintenance, while clients may not have this capability. The model may be dynamically generated / updated at the server by exploding / imploding details on specific and available services depending on the interaction request from the client. With the server having the ability to either provide, or refer to a provider of, such a service, the client will be able to assimilate information about the server and hence act either pro-actively or reactively. This ability to interact and coordinate allows the embedding of domain-specific intelligence within client applications.

For example, assume that this technique is been used to build an automated factory floor. For simplicity, assume that each factory floor resource, would therefore, in essence be designed as an ICICLE. Further assume that a resource, say R_1 , at the beginning of a job flow is stuck on some job. With its communication-centric structure, R_1 is able to communicate this information to other members of the group, if it chooses to do so. Correspondingly, other members of the group can reevaluate their goals / strategies on spending their resources more productively by reacting intelligently to this information. *However, in designing such a system, the first order of priority is to devise a mechanism to discern the information to be shared!* To accomplish this, as part of the development process, we propose to use a Petri-net based analysis technique to analyze each ICICLE modeled by a Petri Net model and correspondingly extract the "information" to be shared using PN invariants.

An ICE element must be able to readily perceive, appreciate, understand and demonstrate cognizant behavior. For this reason, a design method must first capture the important attributes of the *functionality* of the ICE element in its representation. In this regard, identification and handling of failures is of critical importance. A software modeling mechanism must therefore be flexible and sufficiently rich, to allow a designer represent and understand local failure behavior in detail. In our work, we use Hierarchical Time-Extended Petri Nets, or H-EPNs, [3] a hierarchical Petri-net modeling mechanism with the capability for selective expansion / contraction of subnets to model selective information sharing [1]. The base model for each ICICLE is derived by a bottom-up synthesis of smaller Petri net models, each of which pertains to an atomic (i.e., a well-defined functionality such as the request and processing of a connection, a robot move operation, etc.) functionality of the ICICLE. Composing these models into one higher-level ICICLE model is accomplished by traditional Petri-net synthesis techniques. Models can be synthesized many times over to create a nesting of sub-models, called subnets. Depending upon the level of information-sharing detail these subnets can in real-time be either collapsed in

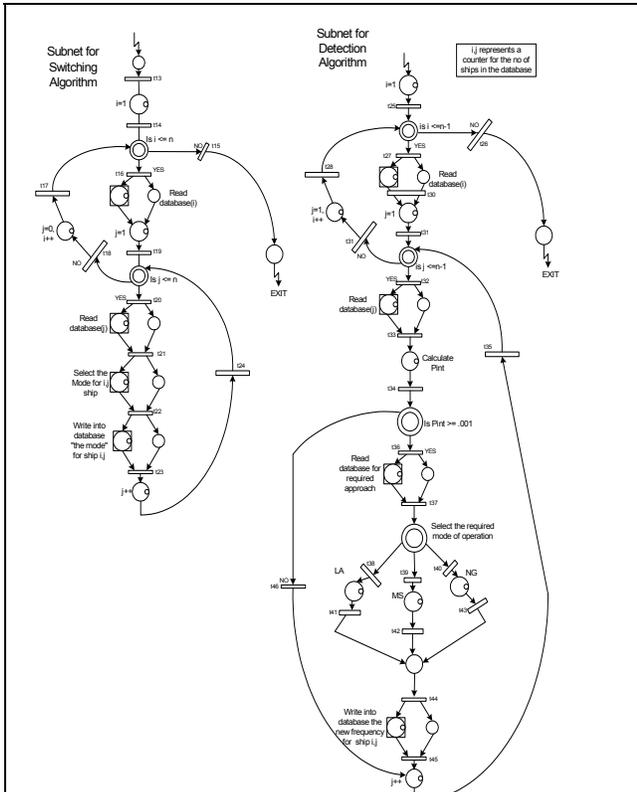


Figure 4: Switching and Detection Structure

Figure 4. PN Subnet Models for Interference Detection and Protocol Switching

frequency interference during operation. For normal and proper operation of these radar units, it is imperative that interference-free frequency is available for each radar unit. Whenever interference occurs between radar units, a suitable mechanism needs to be adopted to resolve this interference. This detection and resolution process that is associated with these affected radars is of extreme importance in time-critical situations.

Since, these radar units work in a dynamically changing environment, such as frequent change in direction, frequent change in coordinates of the ship and other such parameters that affect the frequency of operation, a need for automated negotiation for interference deduction and resolution between the affected radar units and a choice of an appropriate coordination framework is most desirable.

Two important steps that have major impact in this automated negotiation for interference detection and resolution are (i) The process of determining interference, and, (ii) Resolution process of this interference.

These two steps play very critical role in the proper and intelligent working of the naval radar units. Hence, due to time-restrictions only these two steps have been studied and modeled. The first step, basically deals with the actual process of interference deduction. Two ships are said to

have interfering frequencies when their probability-of-interference is greater than 0.001. This probability of interference P_{int} is a quantitative approach based on sound communication theory principles that tells us if interference can occur between two radars based on a set of parameters. So, the detection algorithm module determines (by reading the database for each pair of ship in the group) if the P_{int} value is greater than 0.001 and if so, switches to a particular mode of operation that resolves this interference. This mode of operation is a pre-computed mode for each ship based on the communication parameters such as position, direction etc. For more details, readers can refer to [5]. The second step involves the process of determining the mode of operation for each ship. The modules, switching algorithm and mode selection does exactly this. It reads the parameters for each pair of ship in the group from the database, and determines the mode of operation. This mode of operation is then written in to the database for that particular pair of ship. Since, this kind of an application envisaged a real-time application development process; hence, this problem was taken as a specific case study to develop a Petri Net model and show that analysis of Petri Nets does produce useful and important results for software design and testing. Begin a real time application simulation; hence it became imperative to build a good design (that is robust and scalable) and to build “necessary” test cases for testing.

Currently we are working on two issues, namely: (i) building a Dynamic Switching Protocol (DSP) for switching between the three existing approaches based on

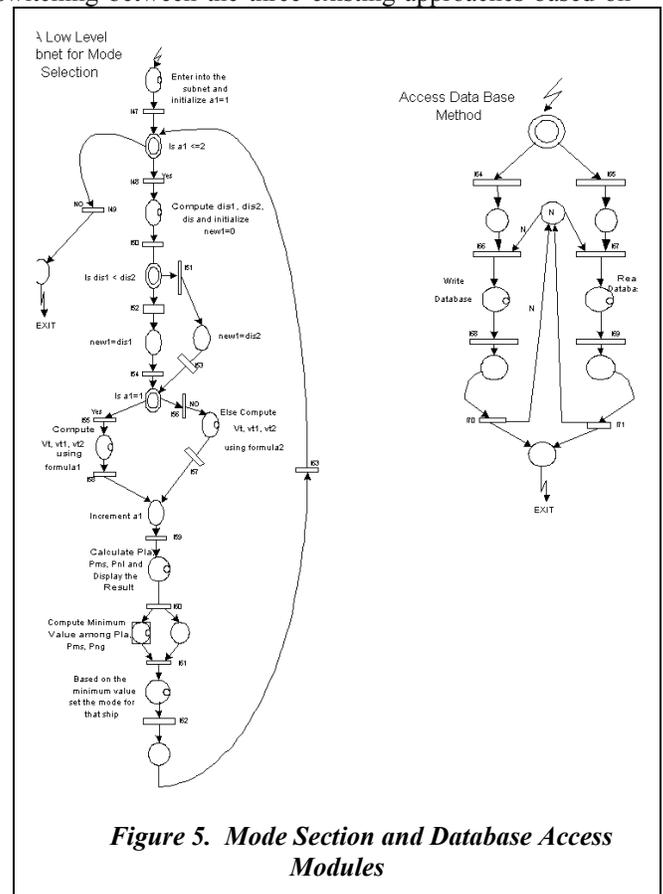


Figure 5. Mode Section and Database Access Modules

various conditions and simulated information. (ii) Developing advanced negotiation protocols that incorporate the advantages provided by the different modes in different situations. These protocols also allow several ships to negotiate at any given instant. Given this background, it is but appropriate to understand (i) the design of the system with respect to information shared between the various subsystems; and (ii) the “necessary” testing to be performed on the simulation system before a fieldable prototype is built. The block diagram of the simulation software system is shown in Figure 2 and the Petri Net model is presented in Figure 3.

The subnets for the interference detection and protocol switching algorithm are presented in Figure 4. The first subnet in Figure 4 is used for switching purposes. It reads the database for every pair of ship and for each pair, a

Table 1. Minimal Invariants for the Various Subnets

Subnet Name	T-Invariants
Switching Algorithm (sa)	X1: t13 t14 t15 t1sa X2: t16 t19 t18 t17 X3: t20 t21 t22 t23 t24
Detection Algorithm (da)	X1: t25 t26 t1da X2: t27 t29 t30 t31 t28 X3: t32 t33 t34 t36 t37 t38 t41 t44 t45 t35 X4: t32 t33 t34 t36 t37 t39 t42 t44 t45 t35 X5: t32 t33 t34 t36 t37 t40 t43 t44 t45 t35 X6: t32 t33 t34 t46 t35
Mode Selection (ms)	X1: t47 t49 t1ms X2: t48 t50 t52 t54 t55 t58 t59 t60 t61 t62 t63 X3: t48 t50 t51 t53 t55 t58 t59 t60 t61 t62 t63 X4: t48 t50 t51 t53 t56 t57 t59 t60 t61 t62 t63 X5: t48 t50 t52 t54 t56 t57 t59 t60 t61 t62 t63
Access Database (adb)	X1: t64 t66 t70 t72 t1adb X2: t65 t67 t71 t73 t1adb
Minimum Value (mv)	X1: t72 t74 t79 t81 t1mv X2: t72 t73 t76 t78 t1mv X3: t72 t73 t77 t80 t1mv X4: t72 t74 t75 t76 t78 t1mv

mode is computed depending upon various ship and radar parameters. The other subnet in Figure 4 (on the right) is used for interference detection. In this subnet, each pair of ship is selected from the group. Based on its parameters such as frequency, position and other factors, the probability of interference is computed. If this value, Pint, is greater than a predetermined value (0.001), frequency interference is said to take place. So, in order to resolve this interference, a mode of resolution is selected from the database and dynamically switched to operate the radar units in one of the 3 possible modes namely, LA, MS and NG. This is process is then repeated for all pairs of ships in the group.

The first subnet in Figure 5 illustrates the actual mode computation process between two interfering agents. For

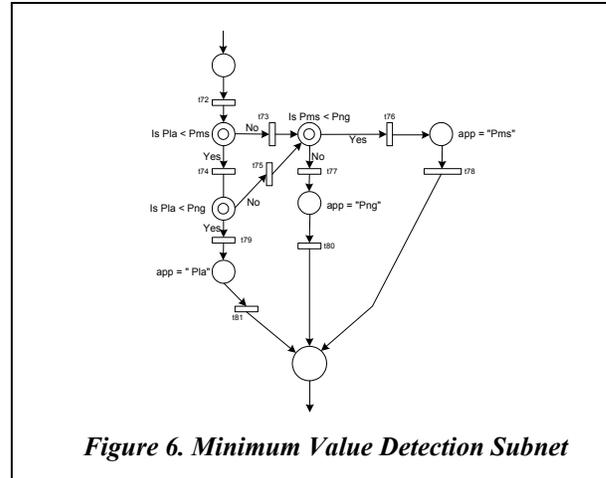


Figure 6. Minimum Value Detection Subnet

each interfering agent, Pla, Pms and Png (P is an evaluation Index for various modes) are determined which are based on sound principles from communication. Whichever value is less among Pla, Pms and Png that particular mode of operation is selected for that particular ship.

The smallest value is determined by calling a simple minimum value determination subnet, which is illustrated in Figure 6.

IV. DESIGN ANALYSIS AND TEST CASE DESIGN

PN invariant analysis [2] were performed to identify the various minimal T-invariants [4] on all the models generated. These are illustrated in Table 1. Transitions with a ‘1’ followed by the corresponding subnet name

represent loop back transition which have not been shown in any of the Petri Net models. They have been included in the design in order to build test cases for the individual modules and for the invariant analysis.

When a PN is used to represent a software design structure, the T-Invariants obtained from the PN model gives all the independent sub-flows in the design. Based on these sub-flows, software testing can be done on the equivalent design. In order to generate the software test cases, MTCS algorithm [1] is used on the T-invariants generated. The algorithm for the MTCS transition is already presented in [1] and hence not presented here. Depending upon the design requirements of the software designer, the transitions from the invariants can either be refined to obtain the design choice transitions or testing

choice transitions. The methodology used to obtain this refinement is discussed in detail in [1]. The decision associated with these transitions aid us in the design and testing of software. The transitions obtained from the design choice strategy aid in the decision of information hiding and encapsulation. Whereas the decision corresponding to the transitions obtained from the testing choice strategy can be used for building software test cases. These transitions correspond to the inner most conditions being checked in the actual software design. Building test cases on these decisions ensure that all independent sub-flows in the design have been checked or exercised at least once. Thus, building test cases on these MTCS transitions ensures that the equivalent software is being tested for their necessary and sufficient conditions. Applying the MTCS algorithm in [1], the various MTCS transitions were obtained. These are shown in Table 2. The following presents the design and testing analysis of the results obtained.

A. Design Analysis

Switching Algorithm Subnet

The design choice transitions for this subnet are t15, t16, and t20 or t15, t18, and t20. These transitions in the original PN model correspond to different decisions being made. In the first set, transitions t15 and t16 corresponds to check that is made on the value of the variable 'i'. These transitions are fired on the event that i becomes greater than the number of ships in the group (t15) or i is less than or equal to the number of ships in the group. This means that statements after this decision are executed repeatedly for every pair of ship in the group. These statements actually correspond to a particular mode being assigned for every ship pair. So, in the actual design, how this mode assignment is done need not be revealed to other subsystems. The only information that needs to be known is whether all the ships have been assigned a particular mode of operation. From the other set of transitions we can get information as at which ship pair is being currently operated on. Even this decision might be quite crucial or relevant to other subsystems. So, extending this idea to the

OOD, the statements that are executed after this decision (is evaluated to be true) *can* be declared as private methods and the variables 'i' and 'j' could be declared as a public variable.

Detection Algorithm subnet:

For this subnet, the design choice transitions are t26, t27, and t36. Again t26 and t27 transitions correspond to the value of 'i' being checked. That is, inside this subnet the value of 'i' is checked with the number of ships in the group. This decision actually correspond to deciding whether all ship pairs have been formed and if there is an interference among any pair, a suitable mode is switched and interference is resolved. So, this decision might be crucial for other subsystems. How this interference is resolved can actually be abstracted and hence in terms of OOD, those processes that deal with this resolving technique can be declared as private and the variables 'i' and 'j' could be declared as public. Transition t36 is again is a further entry point decision in this subnet that can be clubbed with other higher decisions inside this subnet.

Mode Selection subnet:

The design choice transitions are t48 and t49. The decision that correspond to these transitions is whether the number of interfering agents are less than or equal to two. For each of the interfering agent, a particular mode of operation is determined depending on a set of parameters. So, the decision of finding out whether a mode is been determined for each of the interfering ship might be crucial for other subsystems. So, the variable a1 can be declared as public and the methods and processes that follow this decision can be declared as private.

Access Database subnet:

Here the transitions t64 and t65 correspond to the decision of whether a write is to be made in to the database or whether a read is to be done on a database. This decision of whether the database is in the write state or a read state might be useful for other subsystems. Hence, access data base method might be declared as a public method.

Minimum Value subnet:

Here the transitions are t73 and t74 which correspond to the decision of checking the first value with the second value. Only from this decision can one go on further to find out the smallest value. Since, this subnet might be useful at other places that might need to

Table 2. Various MTCS Transitions

Subnet Name	Greedy Choice Transitions	Design Choice Transitions	Testing Choice Transitions
Switching Algorithm	t13 t16 t20	t15 t16 t20 <i>or</i> t15 t18 t20	t15 t18 t20
Detection Algorithm	t25 t27 t32	t26 t27 t36 <i>or</i> t26 t31 t38 t39 t40 t46	t26 t31 t38 t39 t40 t46
Mode Selection	t47 t48	t48 t49 <i>or</i> t52 t55 t56 t51 t49	t52 t55 t56 t51 t49
Access Database	t64 t65	t64 t65	t64 t65
Minimum Value	t72	t73 t74 <i>or</i> t75 t76 t77 t79	t75 t76 t77 t79

compute the minimum value, hence this method can be declared as a public method.

B. Test Case Design

The test cases derived from the MTCS transitions give all the *necessary* test cases that need to be performed for that particular module or subnet to be deemed error free. On the performance of these tests, the designer can be reasonably confident about the proper functioning of the modules as "independent and self contained". The summary of the test cases for all the modules is shown in Table 3. Various modules were integrated in succession to form larger net models and individual test cases were generated for such modules. Due to the length restrictions of this paper, we are not presenting these results. Interested readers may contact the author for these results.

V. CONCLUSIONS

The ICE-based design approach is (i) a clear and simple approach to design a software system that is distributed, interactive and has the ability to coordinate and communicate with other software, hardware and/or human elements. ICICLES will allow standard implementation patterns as well as facilitate the adoption of reusable software components that increases reliability and product quality. It will help reduce project management costs by keeping project teams focused on delivering improved functionality on well-defined ICICLES.

Unique market requirements force organizations to quickly adapt to the rapid changes. Generic solutions will not always solve particular problems and meet the demands of certain industries. This issue raises the need for solutions that are scaleable, flexible, and can be tailored to the special project needs. Since each ICICLE will contain a mechanism for interaction with other members in the group and a dynamic modeling mechanism to determine "information-sharing" needs, it will provide a flexible framework for designing evolutionary and business software systems.

ICICLES are based on a flexible communication structure with incrementally built coordination structures. Reasoning and other application-specific modules may be built based on domain-specific needs. Thus, software systems can be built as a combination of components that vary in their capabilities ranging from ones that are highly intelligent to those that are worker modules. Moreover, use of a dynamic cooperation model allows for components that can have a time-dependent interaction behavior. In a system with many ICICLES, the presence or absence of a particular component would thus be of little significance as long as the system is built with redundant structures - if a component is present and is willing to cooperate for providing some required service it will be utilized. This

allows the required flexibility to leverage industry and vendor partnerships.

There are a wide variety of issues to be addressed before the viable realization of this technique. Some of the basic issues to be addressed by our research include the following. (i) Should a dynamic PN model be generated and used exclusively at run time or can it be a pseudo-dynamic entity? (ii) Petri-net models are not immune to the state-space explosion problem. If so, what are the performance limitations of the dynamic generation of "huge" Petri net models. Can a tradeoff be achieved at any level either for the maximum number of allowable subnet expansions or for the number of places / transitions in the net? (iii) Most current algorithms to determine T-invariants use a matrix based representation scheme. Such a scheme results in the manipulation of huge "sparse" matrices. Are there better algorithms to determine the minimal T-invariants? (iv) Can we build a certification/reward mechanism for ICICLE coordination? If so, how is this to be done - by static rules or by dynamic derivation of reward units? (v) What is a good measure of ICICLE and system stability? How does coordination affect stability? (vi) Can ICICLES coordinate over geographically distant networks - possibly using the world-wide-web as a backbone? How is performance affected in such a case? We hope that future research into these issues will enable the development of distributed, interactive (in a domain-specific sense) systems with ubiquitous human interfaces and the ability to exhibit intelligent cooperative behaviors. When research into the above issues is completed, it will allow the development of a repeatable, multi-phase process (of designing ICICLES) to yield production-caliber, deployable solutions in the shortest possible time.

References

- [1] S. Ramaswamy, "A Petri net based approach for establishing necessary software design and testing requirements", *Special Session on Petri Nets in Systems Design, IEEE Conference on Systems, Man and Cybernetics*, Nashville, Oct-2000.
- [2] S. Ramaswamy, A. Suraj, K. S. Barber, "An Approach for Monitoring and Control of Agent Based Systems", *Proceedings of the 1997 IEEE International Conf. on Robotics and Automation*, Albuquerque, New Mexico, April 1997.
- [3] S. Ramaswamy, K. P. Valavanis, K. S. Barber, "Petri Net Extensions for the Development of MIMO Net Models of Automated Manufacturing Systems", *Journal of Manufacturing Systems*, Vol. 16, No. 3, May/June 1997, pp. 175-191.
- [4] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol. 77, No. 4, April 1989, pp. 541-580.
- [5] S. Ramaswamy, K. Srinivasan, P. K. Rajan, R. MacFadzean, S. Krishnamurthy, "A Distributed Agent-based Simulation Environment for Interference Detection and Resolution", *Special Issue on Software Agents and Simulation, SIMULATION*, June 2001

Table 3. Recommended Test Cases for the Various Modules

Subnet name	Transition involved	Test Cases	Statement being executed	Importance of this test
Switching Algorithm	t15	01. $i > n$ 02. $i = n$ 03. $i < n$	Exit the subnet	Loop Check
	t18	04. $i < n$ 05. $i = n$ 06. $j < n$	$j=1$ and $i++$	Whether Incrementing of i takes place at the correct values of i and j to account for all ship pairs
	t20	07. $i < n$ 08. $i = n$ 09. $j = n$ 10. $j < n$ 11. $j = n$	Actual mode selection operation	Whether a mode is determine for all ships in the group
Detection Algorithm	t26	01. $i > n$ 02. $i = n$ 03. $i < n$	Exit the subnet	Loop Check
	t31	04. $i < n$ 05. $i = n$ 06. $j > n$	$j=1$ and $j++$	Whether Incrementing of i takes place at the correct values of i and j to account for all ship pairs
	t38	07. $i < n$ 08. $i = n$ 09. $j < n$ 10. $j = n$ 11. $Pint > .001$ 12. $Pint = .001$ 13. $Sel Mode = LA$	Operate in LA mode	Forcing execution to check whether ship pair can operate in the LA mode
	t39	14. $i < n$ 15. $i = n$ 16. $j < n$ 17. $j = n$ 18. $Pint > .001$ 19. $Pint = .001$ 20. $Sel Mode = NG$	Operate in NG mode	Forcing execution to check whether ship pair can operate in the NG mode
	t40	21. $i < n$ 22. $i = n$ 23. $j < n$ 24. $j = n$ 25. $Pint > .001$ 26. $Pint = .001$ 27. $Sel Mode = MS$	Operate in MS mode	Forcing execution to check whether ship pair can operate in the MS mode
	t46	28. $i < n$ 29. $i = n$ 30. $j < n$ 31. $j = n$ 32. $Pint < 0.001$	$j++$	To check whether after each ship pair operation, the next pair is being fetched
Mode Selection	t49	01. $a1 > 2$ 02. $a1 = 2$ 03. $a1 < 2$	Exit the subnet	Loop Check
	t52	04. $a1 < 2$ 05. $a1 = 2$ 06. $distance1 < distance2$	$new1 = distance1$	Computing $new1$ for the first pair of interfering ship when $distance1 < distance2$
	t51	07. $a1 < 2$ 08. $a1 = 2$ 09. $distance1 > = distance2$	$new1 = distance2$	Computing $new1$ for the first pair of interfering ship
	t55	10. $a1 < 2$ 11. $a1 = 2$ 12. $distance1 < distance2$ 13. $a1 = 1$	Compute vt_1, vt_2 with first formula	Computing the values for the first pair when $distance1 < distance2$
	t55	14. $a1 < 2$ 15. $a1 = 2$ 16. $distance1 > = distance2$ 17. $a1 = 1$	Compute vt_1, vt_2 with second formula	Computing the values for the first pair when $distance1 > = distance2$
	t56	18. $a1 = 2$ 19. $distance1 < distance2$ 20. $a1 < > 1$	Compute with formula2	Computing the values for the second ship when $distance1 < distance2$
	t56	21. $a1 < 2$ 22. $a1 = 2$ 23. $distance1 > = distance2$ 24. $a1 < > 1$	Compute values with formula2	Computing the values for the second ship when $distance1 < distance2$
Access Database	Not implemented because actual design used vectors			
Minimum Value	t75	01. $pla < pms$ 02. $pla < png$	$app=Pla$	When Pla is lowest
	t76	03. $pla > pms$ 04. $pla = pms$ 05. $pms < png$	$app=pms$	When Pms is lowest or equal to Pla mode and Png is greater than both these values
	t77	06. $pla < pms$ 07. $pla > png$ 08. $pla = png$	condition $pms < png$ being checked	When Pla greater than Png but less than or equal to Pms
	t79	09. $pla > pms$ 10. $pla = pms$ 11. $pms > png$ 12. $pms = png$	$app=png$	When Png is lowest or equal to Pms and Pla greater than both these values