Available online at www.sciencedirect.com

SCIENCE DIRECT

J. Parallel Distrib. Comput. 64 (2004) 853–865

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

Aspect-oriented design of sensor networks

R.R. Brooks,[a,*] Mengxia Zhu,[b] Jacob Lamb,[a] and S.S. Iyengar[b]

[a] *Distributed Systems Department, Applied Research Laboratory, The Pennsylvania State University, P.O. Box 30, State College, PA 16804-0030, USA*
[b] *Department of Computer Science, Coates Hall, Louisiana State University, Baton Rouge, LA, USA*

Received 20 June 2003; revised 7 December 2003

Abstract

The rapid technology development in wireless communication and embedded micro-sensing devices has made the distributed sensor networks (DSN) an area of national importance. Wireless sensor networks are an important military technology with civil and scientific applications. More importantly, the design and analysis of sensor networks can be quite complicated, since each node must simultaneously interact with many other nodes to achieve multiple goals. In this paper, we show how this problem can be made tractable by designing separate protocols for each aspect of a node's behavior. We model this discrete event system by Petri Nets and then formulate three aspect hierarchies: sensing, communications, and command. Within each aspect hierarchy, a node is dynamically assigned roles. To combine the hierarchies, control specifications are derived that enforce consistency across the aspects. Controllers are created using three discrete event methodologies to show how computationally independent aspect-oriented designs can be integrated to form a unified distributed system. The controller methodologies used are: (i) Petri Nets, (ii) finite state automata (FSA) using the Ramadge and Wonham approach, and (iii) vector addition control using the Wonham and Li approach. Finally, we contrast the controller design methodologies by presenting the advantages and disadvantages for each method. In conclusion, for our Petri Nets modeled DSN system with n places and m transitions, constructing Petri Nets controller is computationally efficient but with controller execution time complexity of $O(n \times m^2)$. On the other hand, FSA controller provides prompt response with time complexity of $O(n \times m)$ at the cost of manual offline state space search and encoding. Thus this method is only applicable to medium and small size system.

Keywords: Distributed sensor networks; Aspect oriented design; Surveillance; Discrete event control; Petri Nets

1. Introduction

Advances in micro-electro-mechanical systems (MEMS) technology and wireless communication have spurred the development of wireless sensor networks consisting of a large number of low cost, low power, and collaborating sensor nodes [2]. Since the positions of sensor nodes are not pre-determined, and nodes are prone to frequent failure, system must self-organize and maintain the topology of its internal control structures. As the number of sensor nodes becomes large, human management becomes infeasible and self-organization becomes critical. Moreover, the regions under surveillance are subject to unpredictable environmental disturbances, requiring adaptation by the system.

We derive models of, and controllers for, distributed sensor networks consisting of multiple cooperating nodes. Each battery-powered node has wireless communications, local processing capabilities, data storage, and limited mobility. We derive hierarchical structures that support user control of the distributed system. Of particular interest is self-organization technology for adapting the system to the user's needs.

Many aspects need to be considered simultaneously when designing distributed surveillance networks. Firstly, some human guidance is needed to determine global tasks that the network should perform. This is especially important for military applications, where rules of engagement may change abruptly. Secondly, sensor nodes need to maintain communications connectivity, even in the presence of non-trivial disturbances. Thirdly, the system must efficiently collect and

*Corresponding author. Present address: Department of Electrical and Computer Engineering, Clemson University, P.O. Box 340915, Clemson, SC 29634–0915, USA. Fax: +1-814-863-1396.
E-mail addresses: rrb@acm.org (R.R. Brooks), iyengar@bit.csc.lsu.edu (S.S. Iyengar).

0743-7315/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2003.12.003

interpret sensor readings. Each aspect is quite complex, requiring coordination among nodes. This coordination requires different nodes to have different roles. Self-organization requires these roles to be chosen at run-time.

The complexity of this system is non-trivial. The task of designing the system could easily become intractable. In recognition of this, we divide network control into three hierarchies: (i) operational command, (ii) network communication, and (iii) collaborative sensing. Each hierarchy considers one aspect of the network's operation. This divide and conquer approach simplifies the system design task. Each aspect hierarchy maintains a control structure suited to its goal and evolves independently of the others.

Each aspect hierarchy has a hierarchical structure with three roles: (i) leaf node, (ii) cluster head and (iii) root node. We will explain how these roles create command structures of an arbitrary number of levels for scalability, reliability and energy conservation. Each node fulfills multiple roles. Roles are chosen dynamically and can change during the course of a mission.

Aspect-oriented design has flexibility, but also has associated risks. Since three hierarchies control each node simultaneously, inconsistencies due to conflicting aspect goals need to be resolved. We use discrete event control methods to integrate the hierarchies and resolve conflicts. In addition, discrete event control allows us to verify the integrated system for deadlocks, livelocks, and uncontrollability.

The remainder of the paper is organized as follows. Section 2 gives a review of Petri Nets. Section 3 describes the structure of the network hierarchies. In Section 4, we provide control specifications. The controllers are derived in Section 5. Section 6 provides results from simulation run using the controllers. Section 7 is the conclusion.

2. Petri Nets

Petri Nets are a graphic mathematical model for describing information flow introduced by Carl Adam Petri's dissertation in 1962. The model proved versatile in visualizing and analyzing the behavior of asynchronous, concurrent systems. Later research led to the direct application of Petri Nets in automata theory. Petri Nets model the relationship between events, resources, and system states [13].

A Petri Net is a bi-partite graph with two classes of nodes: *places* and *transitions*. The number of places and transitions are finite and non-zero. Directed arcs connect nodes. Arcs either connect a transition to a place or a place to a transition. Arcs can have an associated integer weight. DEDS state variables are represented by places. Events are represented by transitions. Places contain *tokens*.

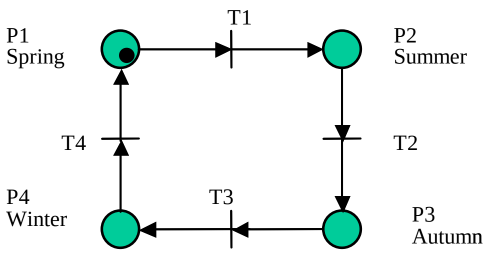

Fig. 1. Petri Net model of the cycle of the seasons with four possible markings: {1000,0100,0010, 0001}.

The DEDS state space is defined by the *marking* of the Petri Net. A marking is a vector expressing the number of tokens in each place. A transition is enabled when the places with arcs incident to the transition all contain at least as many tokens as the weight of the associated arcs. The firing of a transition removes tokens from all places with arcs incident to the transition and deposits tokens in all places with arcs issuing from the transition. The number of tokens removed (added) is equal to the weight of the associated arc. The firing of a transition thus changes the marking of the Petri Net and the state of the DEDS system.

Mathematically, a Petri Net is represented as the tuple $S = (P, T, I, O, u)$ with P: Finite set of places, T: Finite set of transitions, I: Finite set of arcs from places to transitions, O: Finite set of arcs from transitions to places and u is an integer vector representing the current marking. Fig. 1 is a Petri Net modeling the cycle of seasons. To design our FSA controllers, we derive Karp–Miller trees from the Petri Nets [4]. Despite their name, Karp–Miller trees are graph structures; they represent all possible markings a Petri Net can reach from a given initial marking.

3. Hierarchy models

3.1. Overview and terminology

In an effort to thoroughly describe the functionality of a remote, multi-modal, mobile sensing network three aspects of system behavior must be addressed:

- Network communication—maintaining communications within the network.
- Collaborative sensing—coordinating sensor data interpretation.
- Operational command—assigning resources within the network and controlling internal system logistics.

Each aspect hierarchy has three roles:

- Root—is the top level of the hierarchy. It coordinates among cluster heads and provides top-level guidance.

- Cluster head—coordinates lower level controllers and propagates guidance from the root to lower layers.
- Leaf—performs low-level tasks and executes commands coming from the upper layers.

In this paper, we provide a Petri Net plant model for each role of each aspect. The Petri Net models of the aspects are given in the appendix.

We have identified global consistency issues that require the nodes to constrain the actions taken by individual aspect hierarchies. These requirements were captured as control specifications and used to derive the appropriate control structures.

Fig. 2 shows the hierarchical relationship between the three aspect roles. To make the hierarchy adaptive, a cluster head can control any number of leaves. Similarly, a root node can coordinate an arbitrary number of cluster heads.

While there are three roles within each aspect hierarchy, the design does not limit hierarchy instances to only three levels. Networks that cover large regions, or operate in highly cluttered environments require more complex organizations. For this reason, internal nodes are inserted between the root node and cluster heads. In our implementation, internal nodes are either root or cluster head nodes that are connected recursively. This allows complex structures to arise as required by the mission.

In the network communication and collaborative sensing hierarchies, the internal nodes are root nodes. A network containing four levels would consist of a number of three-level subnets, each supervised by a root node. Root nodes at the third tier would each in turn report subnet statistics to an overseeing "master" root at the fourth tier. The master root would manage each of the three-level subnets according to subnet capacities. In other words, collections of cluster heads are subnets controlled by a root node. Combinations of cluster heads and root nodes can be controlled by another root node. In this manner the network may be expanded to manage an arbitrary level of complexity.

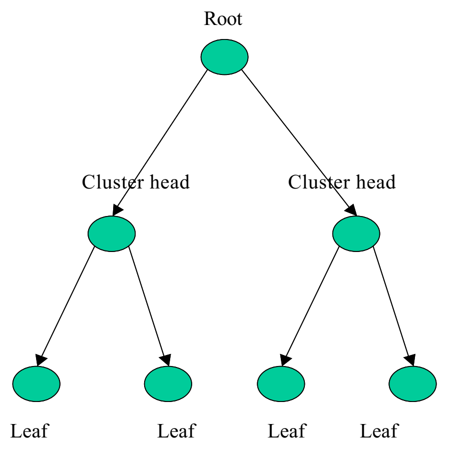

Fig. 2. Relationships between three nodes levels.

Recursion in the network communication and collaborative sensing hierarchies takes place at the root node; however, for the command and control hierarchy recursion takes place at the cluster head. As discussed previously the network communication and collaborative sensing network hierarchies are designed in a fashion in which supervising nodes at each level oversee the activities of subnets. This differs from the operational command hierarchy, where the top level of the hierarchy must be designed as a supervisor overseeing the network as opposed to a subnet. The mapping functions as well as topology maintenance require specific methods be implemented at the tier charged with overseeing the entire network. For this reason, the recursion in the operational command hierarchy is implemented at the cluster head level, the highest level in the hierarchy based on a supervisor–subnet philosophy. The root node controls a set of cluster heads. Cluster heads can coordinate leaf nodes and/or other cluster heads. The independent design and implementation allows recursion in different hierarchies to be designed at different tiers without complications.

A given physical node has a role in each of the three aspect hierarchies. It is important to note that a nodes role in one aspect hierarchy is completely independent of its role for the other two aspects (e.g., a node can be root for the communication aspect, cluster head for the command and control aspect, and leaf for the collaborative sensing aspect). This allows maximum flexibility in network configuration and provides the network with the ability to configure sensing clusters dynamically in order to best process sensor readings from an individual target. Each aspect protocol includes protocols for nodes to dynamically negotiate role changes within the hierarchy (Fig. 3).

3.2. Operational command

The operational command aspect is responsible for maintaining the core functions of the network. The combined operational command aspect hierarchy controls allocation of nodes to surveillance regions, including mapping unknown territory and discovering obstacles. It also controls node deployment, and retreat. Fig. 6 of the appendix demonstrates the interaction between the root, cluster heads, and leaf nodes.

The operational command hierarchy addresses all issues related to network deployment. In general, the topology or geography of the surveillance region is unknown. The operational command hierarchy includes mapping unknown terrains and algorithms for efficient deployment. In addition to initial mapping, a mobile network must update its global map as inaccuracies or changes are discovered.

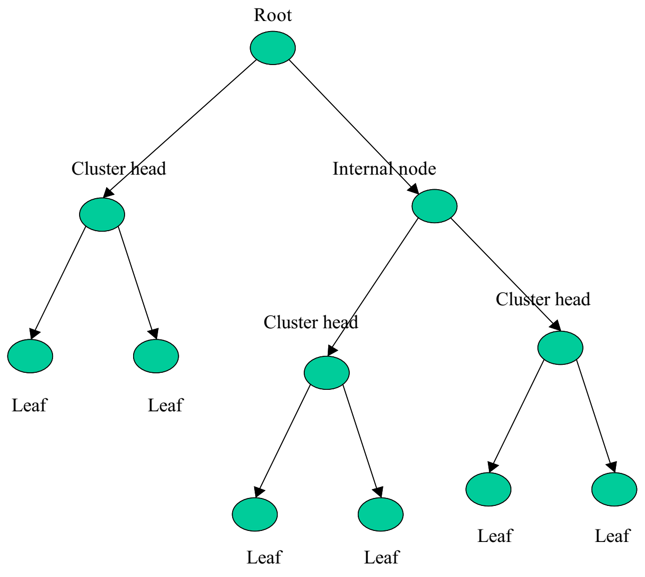

Fig. 3. Example of a more complex structure.

The network reconfigures itself as priorities change. Initial node deployments concentrate nodes in regions: (i) where it is assumed enemy traffic will be heavy or (ii) which are of strategic interest to friendly forces. Over times the network finds areas where enemy traffic is actually flowing, which are likely to differ from those initially anticipated. The strategies of friendly forces will also change over time.

The root node manages network resources and oversees the following functions: mapping the region of interest, node assignment, node reallocation, network topology and network retreat. The root provides information about these functions to the end user and distributes user preferences and commands to appropriate subnets. A pictorial description of the root node is provided in the upper portion of Fig. 6.

Cluster heads (Fig. 6 middle) manage the activities of subnets of leaf nodes and other cluster heads, generate topology reports, interpret commands from the root, calculate resource needs, and monitor resource availability.

Leaf nodes (Fig. 6 bottom) only consider the area they are currently monitoring and retain no global information. Each leaf node directly interacts with its environment, performing terrain mapping and providing position and status information as required by upper levels of the hierarchy.

3.3. Network communications

Network communications is a critical aspect of any mobile network. The network must maintain data flow in the presence of environmental interference, jamming and node loss. At the highest level, the network must retain contact with the user. Throughout the hierarchy, actions include adjusting transmission power, frequency hopping schedules, ad-hoc routing, and movement to correct interference. The combined Petri net models in Fig. 8 of the appendix describe how and when these actions are taken.

Messages within the network may be in one of three classes. The first message class is intact. These messages are not corrupted. The second type of message is corrupted. They must be filtered or retransmitted before proper processing is possible. The final type of message does not reach the intended target at all. A special case exists when the recipient is unaware of the transmission and cannot initiate corrective actions.

Key pathologies, which contribute to message degradation, include transmission range, multi-path fading, background noise and jamming. Symptoms of these pathologies can be used to indicate how the network should react. Detection of multi-path fading should be followed by a pattern of small movements designed to minimize the effect without drastically exchanging the network topology. Background noise may be overcome by modifying the frequency hopping patterns as certain frequencies are less susceptible to certain forms of noise. Weak signals resulting from excessive distance between nodes can be overcome by increasing transmission power.

Beyond correcting packet corruption, the network must also recover from packet losses. To ensure connectivity between nodes and their immediate supervisors, messages passing information up the hierarchy have acknowledgments. If an acknowledgment is not received, retransmission occurs according to parameters set by end users. When retransmissions are exhausted, a supervisor may have to be replaced. When communications with a supervisor is severed, leaf nodes (Fig. 8 bottom) and cluster head nodes (Fig. 8 middle) immediately enter a promotion cycle. The node waits for an indication that a replacement supervisor has been chosen. If none is received, the node promotes itself to the next level. It broadcasts that it has assumed control of the subnet and overtakes supervisory responsibility. If the previous supervisor rejoins the subnet, it may demote itself.

Lost contact between the root node (Fig. 8 top) and the user is more difficult to address. Upon exhausting retransmissions, the root assumes that contact has been lost and it is isolated from the network. The first action taken is to broadcast a message throughout the network indicating that root contact has been lost. Each node tries to establish contact with the user and become the new root. If this fails, the network is put to sleep by a command propagated down the hierarchy. At this point it is left to the user to re-establish contact. While in this quiescent mode the network suspends operations, and responds only to a wake command transmitted by user.

3.4. Collaborative sensing

The collaborative sensing aspect addresses how well the designated area is observed and how raw sensor data are processed and fused. Coordination of sensor data interpretation is shown in Fig. 7 of the appendix. This aspect hierarchy is based on our sensor network implementation, which was tested at 29 Palms Marine Base in November 2001.

Each node has multiple sensors and may have multiple sensing modalities reducing the node's vulnerability to mechanical failure and environmental noise [3].

Initial processing of sensor information is done by the leaf node (Fig. 7 bottom). Time series data are preprocessed. A median filter reduces white noise and a low pass filter removes high frequency noise. If the signal is still unusable, it is assumed either that the sensor is broken, or that environmental conditions make it impossible, and thus the node temporarily hibernates to save energy. After filtering, sensor time series are registered to a common coordinate system and given a time stamp. Subsequently, data association determines which detections refer to the same object. A state vector with inputs from multiple sensing modalities can be used for target classification [9]. Each leaf node can send either a target state vector or Closest Point of Approach (CPA) event to the cluster head.

Cluster heads (Fig. 7 middle) take care of combining statistics into meaningful track information. A cluster head is selected dynamically. Dynamic cluster head selection chooses the node closest to the signal source (target) as the new cluster head. This guarantees that the cluster head is the one best suited to fuse sensor data with minimal communication overhead.

Root nodes (Fig. 7 top) coordinate activities among cluster heads and follow tracks traversing the area they survey. In this hierarchy, internal nodes are root nodes. They define the sensing topology, which organizes itself from the bottom up. This topology mimics the flow of targets through the system. It has been suggested that this information can guide future node deployment [5].

Network topology reports are calculated by combining computational geometry and graph theoretic techniques. Based on information collected from leaf nodes, cluster heads will generate voronoi diagram and calculate the maximal breach path and best-served path of each cluster and send their topology reports to the root node. The root node generates an overall topology report in a recursive manner. These data define the system topology and the quality of service (surveillance) [10].

4. Control specifications for DSN

Now that we have designed protocols for the behavior aspects of the sensor network and the roles needed for coordination of the tasks performed by that aspect, we need to merge the protocols into a unified system. We do this by considering each aspect protocol a plant model that needs to be controlled by the system. We then synthesize a controller that constrains aspect behaviors to actions that do not violate the needs of the other aspects. Where irreconcilable conflicts exist, the controller decides which aspect of system behavior is most important. We find this a major advance in aspect-oriented design, and the main contribution of this paper. In this section, we derive the control specifications used to integrate the three aspect hierarchies.

Given the set of states G and the set of events Σ, the controller disables a subset of Σ as necessary at every state $g \in G$. Control specifications are defined by identifying state and event combinations that lead the system to an undesirable state. Each specification is a constraint on the system and the controller's behavior is defined by the set of constraints. Control of the DSN requires coordination of all aspects of individual node activities within the constraints of mission goals. Each node has a set of responsibilities and must act according to its capabilities in response.

We identified events that lead to undesirable states. Three primary issues were found that can cause conflicts between system aspects: (i) movement of a node conflicting with the needs of another hierarchy; (ii) nodes attempting to function in the presence of unrecoverable noise; and (iii) retreat commands from the command hierarchy should have precedence over all other commands. Following is the set of constraints the controllers impose on the DSN:

OC—operational command,
CS—collaborative sensing,
NC—network communication.

1. When a node is waiting for on-board data fusion, it should be prevented from moving by NC, OC and CS. Also it should not be promoted by NC or by CS until sensing is complete.
2. Hibernation induced by unrecoverable noise or saturated signal in CS should also force the node to hibernate in NC and OC (and vice versa, for leaf nodes only). Wakeup in CS needs to send wake-up to OC/NC.
3. While the cluster head is in the process of updating its statistics, its leaves should be prevented from moving by NC, OC, or CS.
4. While a cluster head node is receiving statistics from its leaf nodes, it should be prevented from moving by NC, OC, or CS.
5. When sensor nodes are in low power mode as determined by NC, or damaged mode as determined by OC, they should be prohibited from any moving for prioritized relocation or occlusion adjustments.

6. Retreat in OC should supercede all actions, except propagation of retreat command.
7. Nodes encountering a target signal in the CS should suspend mapping action in OC until sensing is complete.
8. Move commands in OC/NC should be delayed while node is receiving sensing statistics from lower levels in the hierarchy.

A control specification is specified as: $l\mu \leqslant b$. l is a $k \times n$ matrix (number of control specifications by the number of places in the plant); μ is an $n \times 1$ matrix representing number of tokens in each place of the plant. b is a $k \times 1$ integer matrix each element representing the total maximal allowed number of tokens in any combination of places.

5. Controller design for DSN

Several controller design methods exist. Each enforces constraints in its own way. Vector controllers use state vector comparison to determine the transitions that violate the control specifications. Petri Nets controllers use slack variables to disable the same transitions. FSA controller uses Moore machines to determine which events should be inhibited in terms of current encoded state. In this section, we show how each of these techniques can be used to create controllers that adjudicate between conflicting aspect hierarchies.

Controller design is complicated by the existence of uncontrollable and unobservable transitions. Uncontrollable transitions cannot be disabled; unobservable transitions cannot be detected. When uncontrollable or unobservable transitions lead to undesirable states, the controller design process requires creating alternative constraints that use only controllable transitions.

5.1. *Finite state machine controller*

Verifying system properties, such as safeness, boundedness and liveness, is done using the Karp–Miller tree. It represents all possible states of the system. Ramadge and Wonham described supervisory control of discrete event process using finite state automaton [14]. We generalized their contribution and proposed our own innovations.

Ramadge and Wonham acquire the state feedback map by enumerating all legal states in the FSA together with their binary control patterns. Introducing the Moore machine and state encoding in our method automatically yields the control pattern from derived logical expressions in terms of their current state.

First, we trim the Karp–Miller tree to reach a finite state automaton as a recognizer for the legal language of the plant. Then, the transition table is used to derive logical expressions in terms of encoded state for each controllable transition [1]. The binary control pattern bit for a particular transition is set to 1 when control specification $l\mu \leqslant b$ continues to hold after the transition firing. For multiple control specifications, the binary control pattern for a particular transition is 1 if and only if the current state satisfies the conjunction of all the inequalities imposed by all constraints.

This approach to FSA modeled controller is unique in two respects. Instead of exploring the algebraic or structural property of a Petri Net as in the case of VDES and Petri Net controllers, it utilizes traditional finite automata to tackle the control problem of discrete event system. In addition, the introduction of the Moore machine to output controller variables guarantees quick response. The quick response is acquired at the cost of extensive searching and filtering of the entire reachable state space offline.

5.2. *Vector addition controller*

The vector discrete event system (VDES) approach represents system state as an integer vector. State transitions are represented by integer vector addition [7]. The VDES is an automaton that generates a language over a finite alphabet Σ consisting of two subsets: Σ_c and Σ_{uc}. Σ_c is the set of controllable events that can be disabled by the external controller. Σ_{uc} is the set of uncontrollable events that cannot be disabled by the controller.

When illegal markings are reachable from the initial marking by passing through a sequence of uncontrollable events, it is an inadmissible specification. Inadmissible control specifications must take an admissible form before synthesizing a controller.

A VDES controller is very similar to a Petri Nets modeled controller. A controller variable c is introduced into the system as a place with the initial value to be b minus the initial value of the transformed admissible control specification [7]. A controllable event will be disabled if and only if its occurrence will make c negative. In our implementation, the controller examines all enabled controllable transitions. If the firing of a transition leads to an illegal state, system rolls back and continues looking for the next enabled transition.

5.3. *Petri Net based control*

Li and Wonham [7,8] made significant contributions to the control of plants with uncontrollable events by specifying conditions under which control constraint transformations have a closed form expression. However, the loop-free structure of the uncontrollable subplant is a sufficient but not necessary condition for control. Moody [11] extended the scope of controller synthesis problems to include unobservable events, in addition to uncontrollable events already discussed in VDES. He also found a method for controller synthesis for plants with loops containing uncontrollable events.

In the Petri Nets controller, a plant with n places and m transitions has incidence matrix $D_p \in Z^{n\times m}$. The controller is a Petri Net with incidence matrix $D_c \in Z^{n_c\times m}$, which contains all the plant transitions and n_c control places. Control places are used to control the firing of transitions when control specifications will be violated. Control places cannot have arcs incident on unobservable or uncontrollable transitions. Arcs from uncontrollable transitions to control places are permitted.

In contrast with the VDES controller, a Petri Nets controller explores the solution by inspecting the incidence matrix. Plant/controller Petri Nets provide a straightforward representation of the relationship between the controller and controlled components. The evolution of the Petri Net plant/controller is easy to compute. In our implementation, the plant/controller Petri Nets incidence matrix is the output that results from the plant and control specification as input [12].

6. Simulation results

6.1. Network simulation (Ns) and network animator (Nam) tools

Ns is a discrete event driven simulation for both wired and wireless network research. Began as a variant of REAL in 1989, Ns has evolved extensively to support most transport protocols such as TCP, UDP, several ad hoc routing protocols, some router mechanisms, and some link-layer mechanisms. Nam is a Tcl/TK-based animation tool for viewing network simulation traces. It enables user to view simulation results such as topology layout and packet flow statistics. Fig. 4 gives a diagram of Ns and Nam.

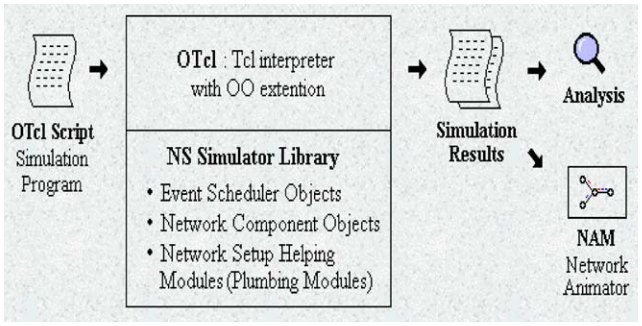
OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

Fig. 4. A schematic view of Ns and Nam [6].

6.2. Example of network simulation result

Initially, there are 10 nodes deployed at different locations in a designated grid, and nine of them range from node 1 to node 9 are displayed as green circles. Node 0 represented as the blue square is the target. All nodes are capable of movement within this grid. Each node is uploaded with a Petri Net as shown in appendix Fig. 9, which is a reduced form of our DSN consisting of three interactive hierarchies.

The 802.11 medium access control (MAC) Layer standard is used here to provide functions that manage and enhance the communications between nodes in a wireless LANs. Our 802.11 MAC Layer uses an 802.11 Physical Layer to carry out the tasks of carrier sensing, transmission, and frames receiving. When a target travels through the grid, nodes that are sensing the target will turn yellow. When the data flow rate of a sensor node is 10 percent or more below that of its neighbors, we say that unfair use of the wireless link is detected, and that node would turn red. Red nodes are supposed to move away from each other to alleviate the unfairness problem.

However, for example, a node could be in the state of sensing while unfairness is detected at the same time. We may ask, should it move or not? According to the control constraints, sensing has higher priority than detecting unfairness. In other words, a node should ignore unfairness detection and remain static while sensing a target. On the contrary, if a node is in the process of moving, it is not allowed to enter sensing state until movement is finished. How could sensor handle these conflicts then?

All conflicts arising from different aspects prove to be effectively mediated by embedded Petri Net controller. Fig. 9 in appendix is a plant/controller compound and is imposed upon each node. Whenever MAC layer detects event, control is conveyed to the Petri Net controller to verify its validness for current system state. Only legal events (transitions) are allowed to happen (fire). Otherwise, system should inhibit corresponding actions.

Ns has been used to simulate the network surveillance with controller imposed. Nam graphically displays the animation results as shown in Fig. 5, two screenshots are captured from the animation.

As seen in the left of Fig. 5, node 3 and node 7 detected unfairness, node 7 moves toward node 3 to alleviate the problem as shown in the right. However, when node 5 was sensing target and then unfairness is detected later on, node 5 turns both red and yellow color. Such nodes are not allowed to move in respond to unfairness, because sensing task has the priority over unfairness. If Petri Net controller is disabled for the same animation, we could see such red/yellow nodes movement. This behavior impairs system performance by not able to properly sense interest target.

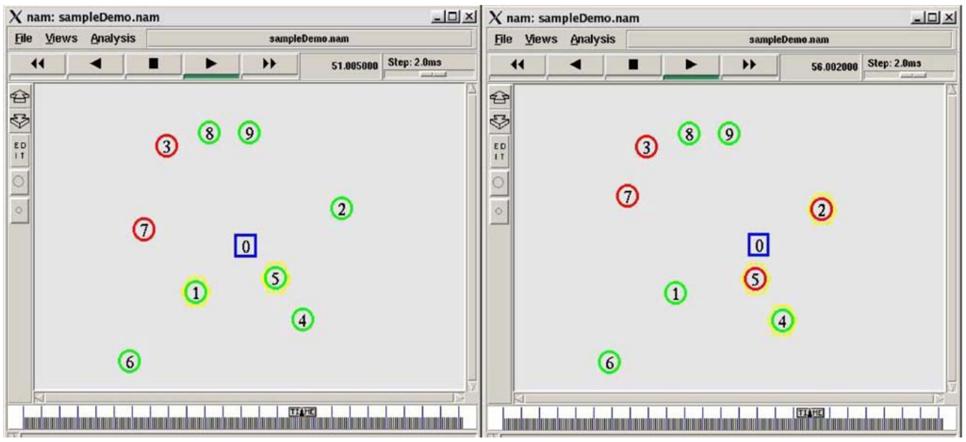

Fig. 5. Snapshots from Nam animation.

On the other hand, if we apply FSA controller to the above example, an offline state-space screening needs to be done manually to get a finite state automaton as a legal language recognizer. This step would incur prohibitively high computation and representation expense for a large-scale system, and makes FSA controller method inappropriate for large-scale system. Then, the mapping between the encoded legal states and the fire-able transitions will be used to derive binary control pattern for controllable transitions. Although it is tedious to construct a FSA controller due to large amount of manual work, it is easy to see that checking the validity for fire-able transitions by evaluating logical expression is faster than doing matrix algebra as in a Petri Nets controller. We assume that almost all places contain zero or 1 token, which is the case for our Petri Nets modeled DSN. In the Petri Net controller, a plant with n places and m transitions has incidence matrix of size $n \times m$, the time complexity needed to check the transition would be $O(n \times m^2)$ by doing matrix algebra [15]. However, for FSA controller, the time complexity is only $O(n \times m)$ by evaluating the logical expression with n Boolean variables for all m transitions. The computation saving is considerable for a large size system with big number of places and transitions.

7. Discussions and conclusion

In this paper, we constructed protocols that defined sensor network behaviors for specific aspects of the network missions. Karp–Miller trees can then be used to verify that each aspect protocol is free of livelock and deadlock [4]. We then analyzed the aspects and found atomic behaviors that were shared by the aspects. In a unified implementation, this could lead to individual nodes receiving conflicting orders. Control specifications were constructed that constrained the aspect behaviors in a manner that enforced over-all mission goals. Discrete event controllers were constructed that enforce these constraints. We showed how this could be done using the three most widely used discrete event control design methods. Lastly, the controller was implemented within a wireless network simulation. Sample runs using the simulator showed that our approach successfully integrated the individual aspects and allowed the system to avoid violating behavior constraints.

Petri Nets, which illustrate process synchronization, concurrent operations, asynchronous events, conflicts and resource sharing, were used to model three aspect hierarchies. The hierarchies modeled the operational command, network communication and collaborative sensing aspects of sensor networks. Each node changes its role in the aspect hierarchy dynamically. The hierarchies evolve independently to best fulfill the constraints of that hierarchy. This independence comes at the cost of conflicts emerging as the hierarchies evolve. Controllers are constructed to resolve these conflicts. We thus achieve design flexibility and simplicity without losing validity.

Through comparison of three controller methodologies, we concluded that all approaches confine system behavior within allowed region, each with pros and cons. Generally speaking, the three approaches can be classified into two categories: FSA belongs to traditional finite automata based controller category and Petri Net modeled and VDES belongs to the Petri Net based controller family.

The traditional Ramadge and Wonham control model is based on a classic finite automaton. Unfortunately, FSA based controllers involve exhaustive searches or simulation of system behavior and are especially impractical for large and complex systems. Offline searching of the entire set of reachable states and the

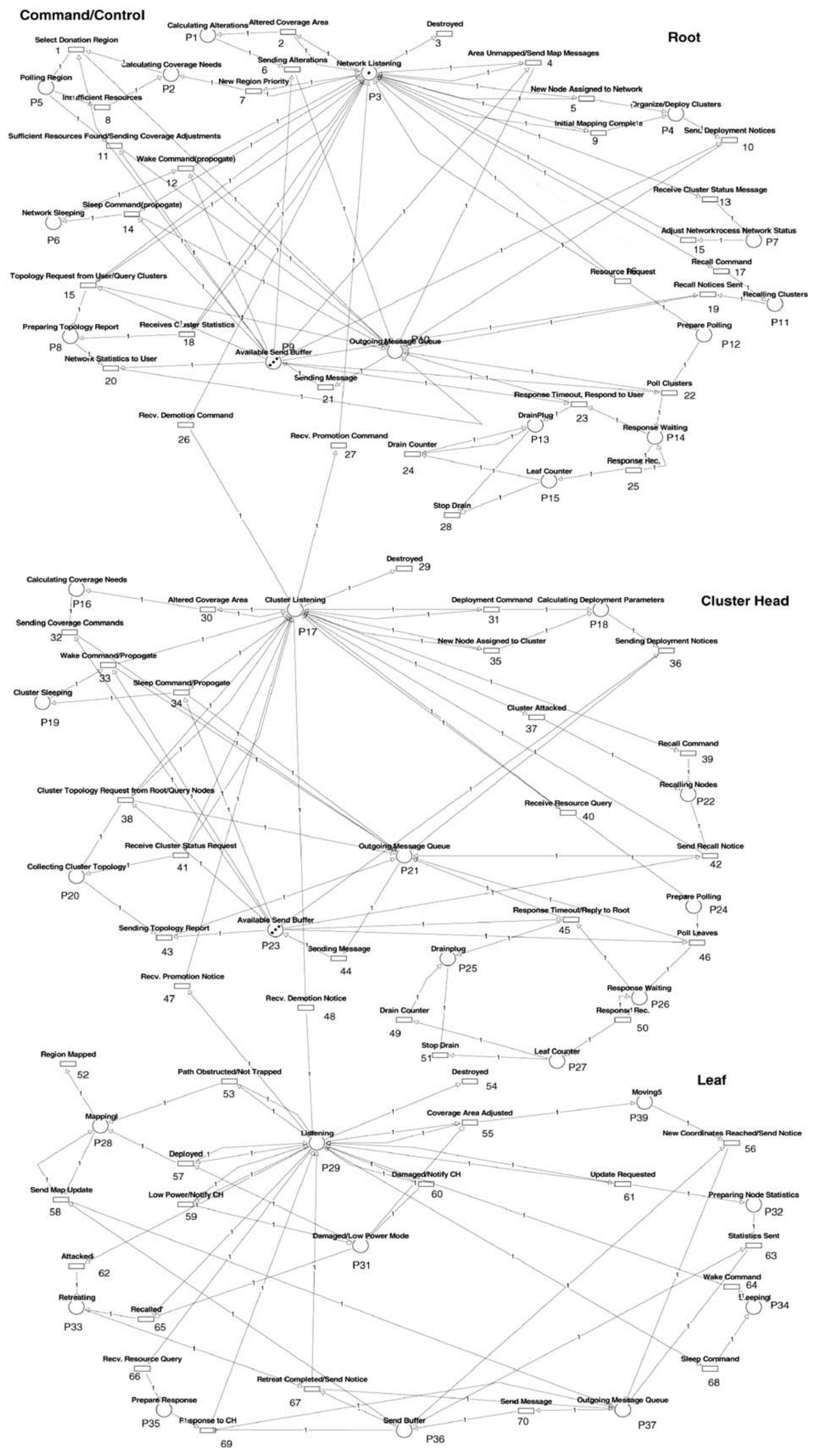

Fig 6. Operational command hierarchy.

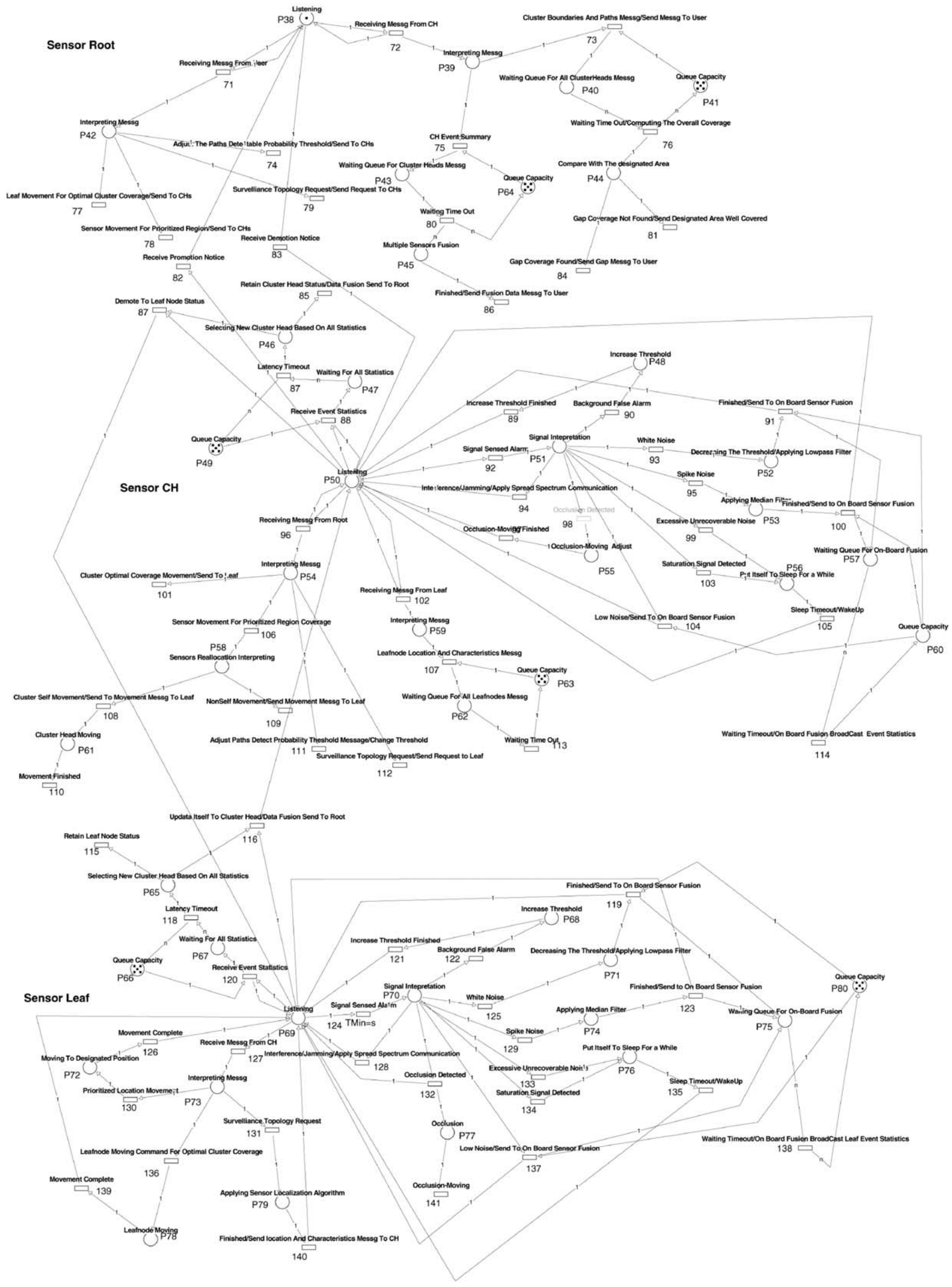
Sensor Root
Listening
P38
Receiving Messg From CH
72
Interpreting Messg
P39
Cluster Boundaries And Paths Messg/Send Messg To User
73
Receiving Messg From User
71
Waiting Queue For All ClusterHeads Messg
P40
Queue Capacity
P41
Interpreting Messg
P42
Waiting Time Out/Computing The Overall Coverage
76
Adjust The Paths Detectable Probability Threshold/Send To CHs
74
CH Event Summary
75
Waiting Queue For Cluster Heads Messg
P43
Compare With The designated Area
P44
Queue Capacity
P64
Leaf Movement For Optimal Cluster Coverage/Send To CHs
77
Surveillance Topology Request/Send Request To CHs
79
Waiting Time Out
80
Gap Coverage Not Found/Send Designated Area Well Covered
81
Sensor Movement For Prioritized Region/Send To CHs
78
Receive Demotion Notice
83
Multiple Sensors Fusion
P45
Receive Promotion Notice
82
Gap Coverage Found/Send Gap Messg To User
84
Retain Cluster Head Status/Data Fusion Send To Root
85
Finished/Send Fusion Data Messg To User
86
Demote To Leaf Node Status
87
Selecting New Cluster Head Based On All Statistics
P46
Increase Threshold
P48
Latency Timeout
87
Waiting For All Statistics
P47
Increase Threshold Finished
89
Background False Alarm
90
Finished/Send To On Board Sensor Fusion
91
Receive Event Statistics
88
Queue Capacity
P49
Signal Sensed Alarm
92
Signal Interpretation
P51
White Noise
93
Decreasing The Threshold/Applying Lowpass Filter
P52
Listening
P50
Sensor CH
Interference/Jamming/Apply Spread Spectrum Communication
94
Spike Noise
95
Applying Median Filter
P53
Finished/Send to On Board Sensor Fusion
100
Occlusion Detected
98
Occlusion-Moving Finished
Excessive Unrecoverable Noise
99
Receiving Messg From Root
96
Occlusion-Moving Adjust
P55
Waiting Queue For On-Board Fusion
P57
Interpreting Messg
P54
Saturation Signal Detected
103
Put Itself To Sleep For a While
P56
Cluster Optimal Coverage Movement/Send To Leaf
101
Receiving Messg From Leaf
102
Sleep Timeout/WakeUp
105
Interpreting Messg
P59
Low Noise/Send To On Board Sensor Fusion
104
Queue Capacity
P60
Sensor Movement For Prioritized Region Coverage
106
P58
Sensors Reallocation Interpreting
Leafnode Location And Characteristics Messg
107
Queue Capacity
P63
Cluster Self Movement/Send To Movement Messg To Leaf
108
NonSelf Movement/Send Movement Messg To Leaf
109
Waiting Queue For All Leafnodes Messg
P62
Cluster Head Moving
P61
Adjust Paths Detect Probability Theshold Message/Change Threshold
111
Waiting Time Out
113
Waiting Timeout/On Board Fusion BroadCast Event Statistics
114
Surveillance Topology Request/Send Request to Leaf
112
Movement Finished
110
Updata Itself To Cluster Head/Data Fusion Send To Root
116
Retain Leaf Node Status
115
Selecting New Cluster Head Based On All Statistics
P65
Finished/Send To On Board Sensor Fusion
119
Increase Threshold
P68
Latency Timeout
118
Waiting For All Statistics
P67
Increase Threshold Finished
121
Background False Alarm
122
Decreasing The Threshold/Applying Lowpass Filter
P71
Queue Capacity
P66
Receive Event Statistics
120
Queue Capacity
P80
Finished/Send to On Board Sensor Fusion
123
Signal Intepretation
P70
Sensor Leaf
White Noise
125
Applying Median Filter
P74
Waiting Queue For On-Board Fusion
P75
Listening
P69
Signal Sensed Alarm
124
TMin=s
Movement Complete
126
Receive Messg From CH
127
Spike Noise
129
Put Itself To Sleep For a While
P76
Interference/Jamming/Apply Spread Spectrum Communication
128
Moving To Designated Position
P72
Occlusion Detected
132
Excessive Unrecoverable Noise
133
Sleep Timeout/WakeUp
135
Interpreting Messg
P73
Saturation Signal Detected
134
Prioritized Location Movement
130
Surveillance Topology Request
131
Occlusion
P77
Waiting Timeout/On Board Fusion BroadCast Leaf Event Statistics
138
Leafnode Moving Command For Optimal Cluster Coverage
136
Low Noise/Send To On Board Sensor Fusion
137
Movement Complete
139
Applying Sensor Localization Algorithm
P79
Occlusion-Moving
141
Leafnode Moving
P78
Finished/Send location And Characteristics Messg To CH
140

Fig 7. Collaborative sensing hierarchy.

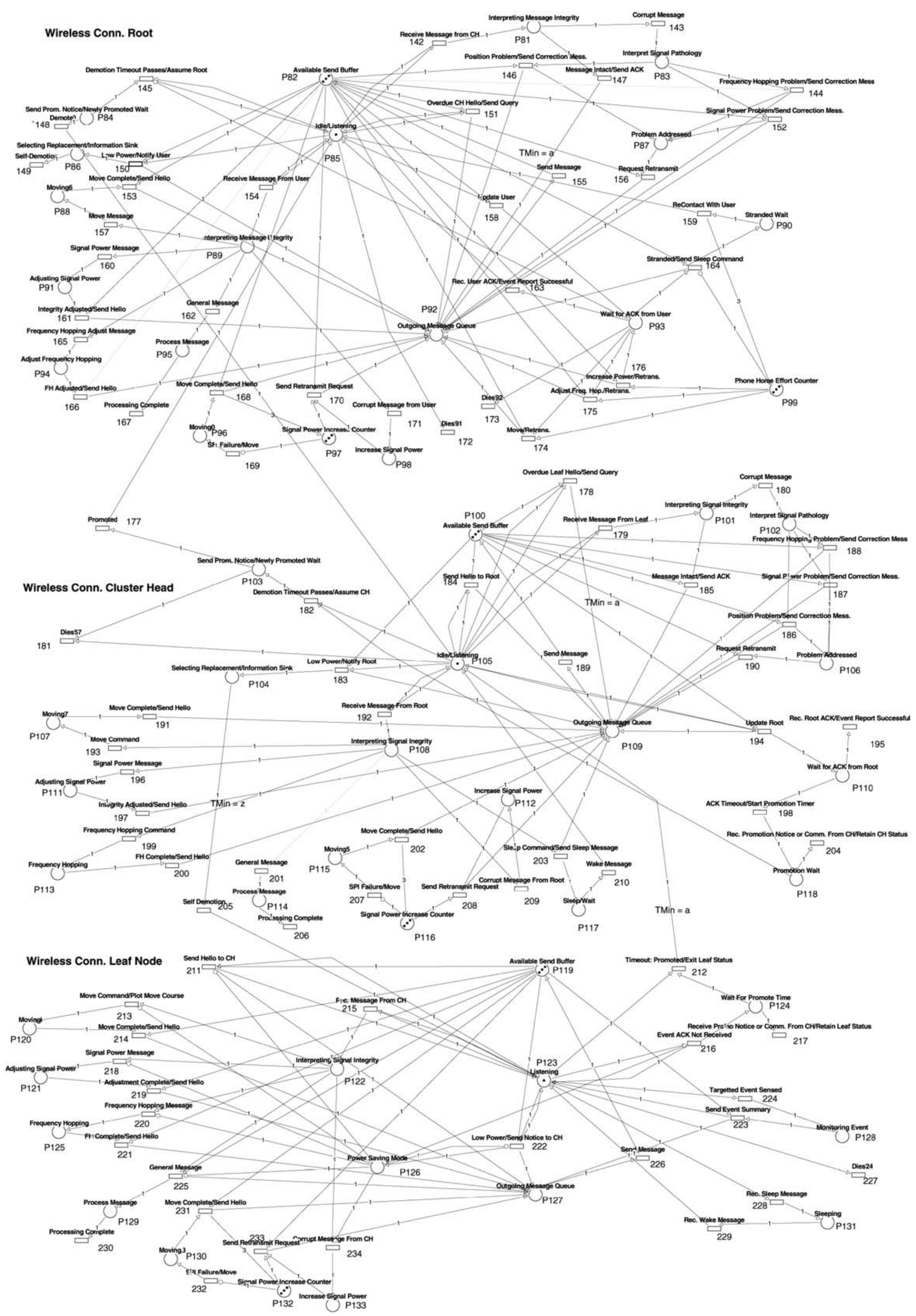

Fig 8. Network communication hierarchy.

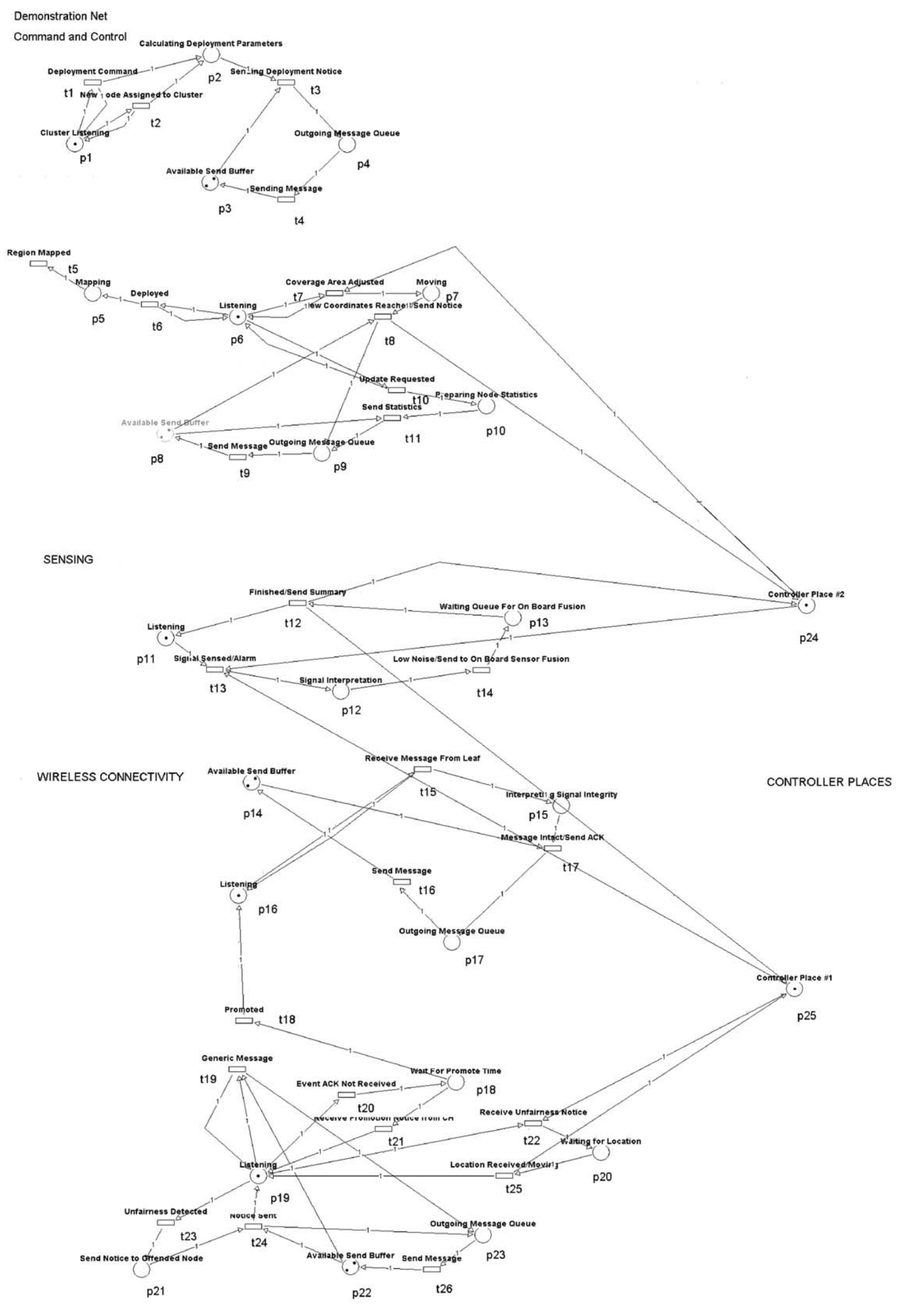

Fig. 9. Sample reduced DSN hierarchy for animation.

direct evaluation of the logical expression assures prompt online response, which is crucial for those systems with strict real-time requirements.

On the contrary, Petri Net based controllers take full advantage of the properties of the Petri Nets. Their efficient mathematical computation employing linear matrix algebra makes controller construction and analysis much easier, but they are still inferior to FSA in the performance of response time. Petri Nets offer a much more compact state space than finite automata and are better suited to model systems that exhibit a repetitive structure. Vector discrete event system controllers explore the maximally permissive control constraint on the Petri Net with uncontrollable transitions by application of the integer linear programming problem, assuming that the uncontrollable portion of the Petri Net has no loops and the actual controller exists [12]. The integrated graphical structure of the Petri Net plant/controller makes system computation and representation straightforward.

Concerns such as execution time and ease of construction and representation can therefore guide decision on which approach to use.

Acknowledgments and disclaimer

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), and administered by the Army Research Office under ESP MURI Award No. DAAD19-01-1-0504. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA), and Army Research Office.

Appendix

Surveillance network Petri Nets plant models (Figs. 6–9).

References

[1] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques and Tools, Addison-Wesley, Reading, MA, 1986.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramanian, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.

[3] R. Brooks, S.S. Iyengar, Multi Sensor Fusion: Fundamentals and Applications with Software, Prentice-Hall Publication Co., Englewood Cliffs, NJ, 1997.

[4] R. David, H. Alla, Petri Nets and Grafcet Tools for Modeling Discrete Event Systems, Prentice-Hall, Englewood Cliffs, NJ, ISBN: 0-13-327537-X, 1992.

[5] B. Deb, S. Bhatnagar, Badri Nath, A topology discovery algorithm for sensor networks with applications to network management, IEEE CAS workshop, September, 2002.

[6] http://nile.wpi.edu/NS.

[7] Y. Li, W.M. Wonham, Control of vector discrete-event systems I-the base model, IEEE Trans. Automat. Control 38 (8) (August 1993) 1215–1227.

[8] Y. Li, W.M. Wonham, Control of vector Discrete-Event System II controller synthesis, IEEE Trans. Automat. Control 39 (3) (March 1994) 512–531.

[9] R.C. Luo, M.G. Kay, Multisensor integration and fusion in intelligent systems, IEEE Trans. Systems Man Cybernet. 19 (5) (September/October 1989) 901–931.

[10] S. Meguerdichian, F. Koushanfar, M. Potkonjak, M.B. Srivastava, Coverage problems in wireless ad-hoc sensor networks, Computer Science Department, Electrical Engineering Department, University of California, Los Angeles, May 2000.

[11] J.O. Moody, Petri Net supervisors for discrete event systems, Ph.D. Dissertation, Department of Electrical Engineering, Notre Dame University, April 1998.

[12] J.O. Moody, P.J. Antsaklis, Petri Net supervisors for DES with uncontrollable and unobservable transitions, Technical Report, ISIS Group, University of Notre Dame, February 1999.

[13] J.L. Peterson, Petri Nets, Comput. Surveys 9 (3) (September 1977) 223–252.

[14] P.J. Ramadge, W.M. Wonham, Supervisory control of a class of discrete event progress, SIAM J. Control Optim. 25 (1) (January 1987) 206–230.

[15] W.M. Wonham, Notes on discrete event system control, System Control Group, Electrical & Computer Engineering Department, University of Toronto, 1999.