Discrete event command & control for networked teams with multiple missions[1]

Frank L. Lewis[2a], Greg Hudas[b], Chee Khiang Pang[a,c], Matthew B. Middleton[a], and Christopher Mcmurrough[a]
and
[a]Automation & Robotics Research Institute, University of Texas at Arlington, Fort Worth, TX, USA
[b]U.S. Army RDECOM-TARDEC, Joint Center for Robotics (JCR), Warren, MI, USA
[c]Department of Electrical & Computer Engineering, National University of Singapore, Singapore

ABSTRACT

During mission execution in military applications, the TRADOC Pamphlet 525-66 Battle Command and Battle Space Awareness capabilities prescribe expectations that networked teams will perform in a reliable manner under changing mission requirements, varying resource availability and reliability, and resource faults. In this paper, a Command and Control (C2) structure is presented that allows for computer-aided execution of the networked team decision-making process, control of force resources, shared resource dispatching, and adaptability to change based on battlefield conditions. A mathematically justified networked computing environment is provided called the Discrete Event Control (DEC) Framework. DEC has the ability to provide the logical connectivity among all team participants including mission planners, field commanders, war-fighters, and robotic platforms. The proposed data management tools are developed and demonstrated on a simulation study and an implementation on a distributed wireless sensor network. The results show that the tasks of multiple missions are correctly sequenced in real-time, and that shared resources are suitably assigned to competing tasks under dynamically changing conditions without conflicts and bottlenecks.

Keywords: Discrete event control (DEC), military battlefield command and control, mission execution and resource assignment, rule-based control.

1. INTRODUCTION

U.S. Army Training and Doctrine Command (TRADOC) Pamphlet 525-66 identifies Force Operating Capabilities required for the Army to fulfill its mission for a networked Warfighter concept. Two such capabilities are Battle Command and Battle-Space Awareness for which there are expectations that networked teams will perform in a reliable manner under changing mission requirements, varying resource reliability, and resource faults. Battlefield or disaster area teams may be heterogeneous networks consisting of interacting humans, ground sensors, and unmanned airborne or ground vehicles (UAV, UGV). Such scenarios should provide intelligent task sequencing for multiple missions, synchronization of efforts for multiple missions, and shared services of resources to augment the capabilities of the remote-site mission commander and on-site war-fighter. This requires a scalable, deployable and mobile networking capability that supports mission tailoring, force responsiveness and agility, ability to change missions without exchanging forces, and general adaptability to changing battlefield conditions.

In this paper we present a computer programmable Command and Control (C2) structure that allows for execution of the decision-making process, control of force resources, and adaptability to change. We describe a rigorous mathematically justified networked computing environment that has the potential to provide the logical connectivity among all team participants including mission planners, field commanders, warfighters, and robotic platforms. Included are data management tools to ensure that the tasks of multiple missions are correctly sequenced in real-time and that shared resources are suitably assigned to competing tasks under dynamically changing conditions.

A rule-based Discrete Event Controller (DEC) has been developed[1] and applied to various engineering and manufacturing applications ranging from schedule planning[2,3,4] to reducing the product life-cycle in prototype designs, as well as Wireless Sensor Networks (WSNs)[5]. The DEC matrix-based formulation[6] is portable and easily implemented on

[1] Supported by ARO grant ARO W91NF-05-1-0314 and the Army National Automotive Center
[2] F. L. Lewis: E-mail: lewis@uta.edu, Telephone: +1 817 272 5972

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **16 MAR 2009**	2. REPORT TYPE **N/A**	3. DATES COVERED **-**
4. TITLE AND SUBTITLE **Discrete event command & control for networked teams with multiple missions**		5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Frank L. Lewis; Greg Hudas; Matthew B. Middleton; Christopher Mcmurrough; Chee Khiang Pang**		5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **US Army RDECOM-TARDEC 6501 E 11 Mile Rd Warren, MI 48397-5000 Automation & Robotics Research Institute University of Texas at Arlington, Fort Worth, TX Department of Electrical & Computer Engineering, National University of Singapore, singapore**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) **TACOM/TARDEC** 11. SPONSOR/MONITOR'S REPORT NUMBER(S) **19665RC**
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release, distribution unlimited**		
13. SUPPLEMENTARY NOTES **Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Defense, Security, and Sensing Symposium, 13-17 April 2009, Orlando, Florida, USA, The original document contains color images.**		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**SAR**	**15**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

any platform. It facilitates industries adapting quickly to fast-evolving market conditions for transition between inventory and products with minimal human intervention[7].

Based on this DEC, we develop here a C2 structure. DEC is easily programmed on a laptop digital computer. DEC is based on *matrices* that contain two types of information. Mission task requirements are prescribed by mission commanders in terms of a Task Sequencing Matrix. This allows commanders to convey purpose without providing detailed direction on how to perform the task or mission. Resource assignments to the tasks are prescribed by field commanders or the warfighter in terms of a Resource Assignment Matrix. As missions change or are added, the task sequencing matrices are easily reconfigured. Multiple missions can be programmed by multiple mission commanders into the same networked team. As resources fail or are added, the resource assignment matrices are easily reconfigured in real-time time.

The discrete event controller (DEC) allows for synchronizing forces and warfighting functions in time, space, and purpose to accomplish multiple simultaneous missions that may change dynamically. The matrix formulation of DEC allows for rigorous mathematical analysis of the performance of the networked team, and reveals problems such as bottleneck resources and shared-resource blocking phenomena. DEC guarantees proper sequencing of the competing tasks of multiple missions, assigning appropriate resources immediately as they become available and resolving conflict situations. DEC can be programmed into networked microprocessors using a novel 'or/and' matrix Boolean algebra that allows programming of rule-based decisions in streamlined software algorithms. Computer or PDA user interfaces can allow automatic generation of the Task Sequencing Matrices given the requirements of mission commanders, and of the Resource Assignment Matrices by field commanders.

2. A DISCRETE EVENT C2 STRUCTURE FOR DISTRIBUTED TEAMS

In this section and the next we describe a command and control (C2) structure for programming multiple missions into heterogeneous teams of distributed agents, and controlling the performance of these missions in real-time. This is a decision-making DEC that contains rules to sequence the tasks in each mission, and to assign resources to those tasks. DEC has a message-passing architecture that is in conformance with Joint Architecture for Unmanned Ground Systems (JAUGS)[10], and is an efficient means to realize the high-level OODA loops (observe, orient, decide, act) of 4D/RCS[11]. DEC is able to coordinate the sequencing of operations between multiple Soldiers and robots efficiently and without conflict, thus contributing to the concept of Safeops.

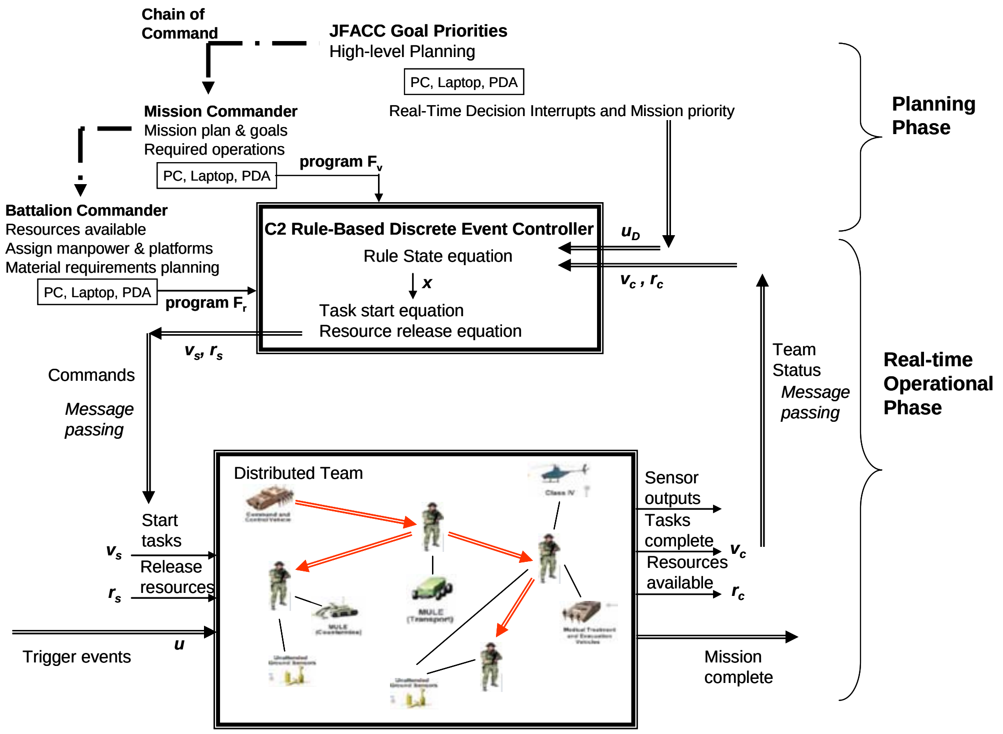


Fig. 1. C&C Rule-Based Discrete Event Controller for Distributed Networked Teams.

DEC allows fast mission programming of distributed teams, and facilitates rapid deployment of man/machine teams, wireless sensor networks, and other event-based systems. DEC provides a seamless C&C architecture that facilitates quickly turning any deployed team into a tactical unit[12]. The DEC runs on a C2 computer and functions as a feedback controller in real-time. See Fig. 1. As a feedback controller, DEC obtains information from each networked agent about which tasks that agent has just completed, and which of its resources are currently available. This information about team status can be transmitted via a message-passing protocol over a wireless sensor network (WSN), or over the internet[5]. Then, given such information from all active nodes, DEC computes which mission tasks *could* be performed next. Then, based on priority measures or war-fighter decision input, DEC decides which tasks the team *should* perform next. Based on this, it sends message-based commands to each agent to perform certain tasks or release certain resources. All this is accomplished efficiently using a computer software DEC tool to be described.

The commands sent by DEC to the team agents could be command inputs into semi-autonomous machine nodes, and could be in the form of messages for decision assistance over a PDA for human agents.

The DEC can be programmed on a digital computer and requires very small code for implementation. The key to the ease of use and implementation of DEC[1] are formal mathematical computations based on *matrices* that contain two types of information (TSM and RAM below), and the use of a nonstandard matrix *or/and* algebra. The functionality of DEC has two phases:

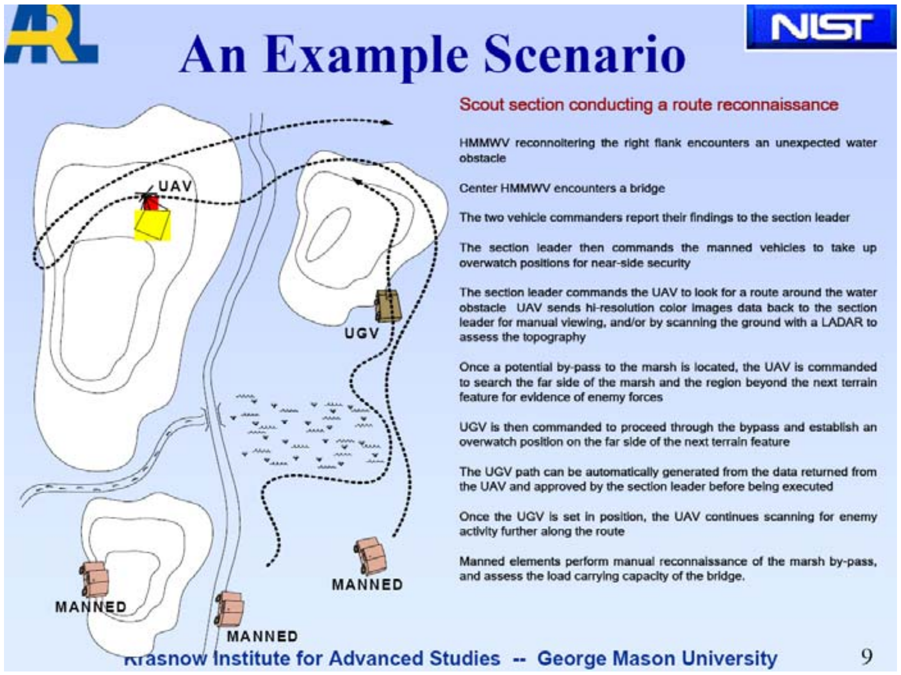
1. *Planning/Programming Phase*, Mission task requirements are prescribed by mission commanders in terms of a Task Sequencing Matrix (TSM). This allows commanders to convey purpose without providing detailed direction on how to perform the task or mission. Next, resource assignments to the missions tasks are prescribed by field commanders or the warfighter in terms of Resource Assignment Matrices (RAM). All this information could be entered via Graphical User Interfaces on laptops, handheld PDA, *etc*. As missions change or are added, these matrices are easily reconfigured in real time. Multiple missions can be programmed by multiple mission commanders into the same networked team that shares the same resources. This is effectively the *world modeling phase* of 4D/RCS[11].

2. *Operational Phase*, the DEC will automatically poll active agents for their status at each event update and properly sequence the tasks of all programmed missions, and assign the required resources. Conflicting requests for resources are automatically handled so as to avoid blocking phenomena. During operation, as resources fail or are added, the resource assignment matrix is easily reconfigured in real-time time to allow uninterrupted mission performance in spite of resource failures. At any time, additional missions may be programmed into the team or deleted.

The Programming Phase is discussed in this section, and the Operational Phase in the next.

2.1 Missions

A mission should achieve desired *goal states* that are triggered by *external events* occurring within the frame of discernment of the distributed team. Mission task requirements are specified by mission commanders in terms of desired responses to special trigger events, and accomplishment of required goals. A mission is defined as a sequence of script behaviors[13], here called tasks, that is triggered by prescribed events and results in prescribed goal states. Mission commanders should be able to convey purpose without providing detailed direction on how to perform the tasks of the mission



An Example Scenario

Scout section conducting a route reconnaissance

HMMWV reconnoitering the right flank encounters an unexpected water obstacle

Center HMMWV encounters a bridge

The two vehicle commanders report their findings to the section leader

The section leader then commands the manned vehicles to take up overwatch positions for near-side security

The section leader commands the UAV to look for a route around the water obstacle UAV sends hi-resolution color images data back to the section leader for manual viewing, and/or by scanning the ground with a LADAR to assess the topography

Once a potential by-pass to the marsh is located, the UAV is commanded to search the far side of the marsh and the region beyond the next terrain feature for evidence of enemy forces

UGV is then commanded to proceed through the bypass and establish an overwatch position on the far side of the next terrain feature

The UGV path can be automatically generated from the data returned from the UAV and approved by the section leader before being executed

Once the UGV is set in position, the UAV continues scanning for enemy activity further along the route

Manned elements perform manual reconnaissance of the marsh by-pass, and assess the load carrying capacity of the bridge.

Krasnow Institute for Advanced Studies -- George Mason University 9

Fig. 2. Sample mission scenario from J. Albus talk, ASC Orlando, Dec. 2008.

[U.S. Army Training and Doctrine Command (TRADOC) Pamphlet 525-66]. A sample mission scenario, taken from[11], is shown in the figure.

2.2 Programming the Missions: Task Sequencing Matrix (TSM)

Given the basic mission requirements specified by mission commanders, a sequence of scripts or tasks that perform the mission is constructed. A grammar-based method for doing this is given[13], where the sequence of tasks for a mission is determined using a planning function such as A-Star search. In 4D/RCS[11] a task analysis is used to create a task decomposition tree. In fact, given basic elemental tasks and the required high-level goals, there are many software planners that can fill in the detailed sequence of steps needed to attain the goals. Notably effective are the so-called Hierarchical Task Network (HTN) planners[2,3,4], which decompose goals into sequences of primitive actions and compound actions, e.g. tasks, that are required to attain those goals. On the other hand, detailed task sequences could also be constructed by aids to the mission commander.

Each mission task has a well-defined initial state and start event, and ends with a well-defined exit[13]. The tasks are fired by rules of the form

Rule i: IF (the tasks required as immediate precursors to task *i* are compete)
AND (the resources required for task *i* are available) THEN perform task *i*

Therefore, each mission can be considered as a sequence of rules prescribing under what conditions each of its tasks can be fired. This is a semi-autonomous rule-base[14] in the sense that it has hard-programmed rules which can nevertheless be interrupted or re-sequenced in real-time by input from the war-fighter.

We define a *mission* as a set of tasks that is triggered by events, and results in a prescribed goal state(s). The mission is prescribed in terms of a *strict partial ordering* of tasks that begins with detected trigger events and ends up with the goal states. A binary relation *P* is a strict partial ordering on a set $\{t_i\}$ if: (1) $(t_i, t_i) \notin P$ (irreflexive), (2) If $(t_i, t_j) \in P$ and $(t_j, t_i) \in P$ then $t_i=t_j$ (antisymmetry), (3) If $(t_i, t_j) \in P$ and $(t_j, t_k) \in P$ then $(t_i, t_k) \in P$ (transitive). We interpret this strict partial order on the set of tasks as specifying the temporal relations between tasks, e.g. $(t_i, t_j) \in P$ if task t_i is required to occur immediately prior to task t_j, e.g., task t_j can only occur if task t_i has just completed.

The intent of this partial order in time is that missions should consist of tasks, some of which should occur when others have just finished, but many of which are not required to be in any definite temporal order with respect to each other (e.g. see Fig. 2). This gives the mission commander great freedom to prescribe only those causal relations between tasks which are tactically important. This allows commanders to convey purpose without providing detailed direction on how to perform the task or mission. Since the missions consist only of partial orderings of tasks, then it will be the responsibility of the DEC to decide the actual ordering of tasks in real time as the events unfold and the resources needed for the tasks become available. The mechanisms for performing this during the Operational Phase are described in the next section.

To capture the mission and its tasks in a convenient and computable form, define the task sequencing matrix (TSM), which has element *(i,j)* equal to 1 if task t_j is a required immediate precursor for task t_i. Note that multiple 1's in a single row *i* indicate that multiple tasks are required as immediate precursors for task t_i. TSM was used by Steward[15], Warfield[16], and Wolter *et al.*[17], and others to sequence the tasks required in manufacturing assembly and part processing.

TSM is a mapping from tasks to tasks. We would like to construct a mathematically formal rule-based DEC that runs as software code on a C2 computer and is capable of sequencing the tasks and assigning resources dynamically in a networked team. In the next section it is shown that this is possible if each task is fired by a *rule*. Therefore, to introduce the rule base, decompose the TSM as

$$TSM = S_v \cdot F_v$$

where the *input TSM* F_v (loosely called simply the TSM) is a mapping from the tasks to the set of rules, and the *output TSM* S_v is a mapping from the rules back to the task space.

This matrix multiplication denoted by the small dot is not the standard matrix multiply, but occurs in the *or/and* algebra, where multiply means *.and.* and addition means *.or.* It is easy to write a programming function to multiply matrices in or/and. This function is part of the simple computational machinery of DEC and is shown in the next section (see Fig. 3).

The input TSM F_v maps the tasks *{t_j}* to a rule base consisting of a set of rules *{x_i}* . It has entry *(i,j)* of 1 if rule *i* requires task *j* as a immediate precursor to its firing. Multiple ones in a row lead to rules of the form

IF (task j_1 and task j_2 and task j_3 have just finished) THEN fire rule i

Output TSM S_v maps the rule base *{x_j}* to the tasks *{t_i}*. It has entry *(i,j)* of 1 if task *i* is to be started when rule *j* fires. Often, there is one rule to fire each task, so that S_v is essentially the identity matrix.

Networked Team Example. To illustrate, a sample mission is shown in Table 1. This mission could be programmed by any user of the networked sensor team. This scenario has a wireless sensor network (WSN) consisting of unattended ground sensors (UGS) and mobile robots (R), possibly UGV or UAV. When the trigger event *u1* (*e.g.* here, a chemical attack) is detected by a UGS (specifically, sensor *UGS1* here), a prescribed sequence of tasks is carried out that includes taking further sensor readings and dispatching mobile robots to gather additional information. The mission ends when sensor *S2* takes a measurement, either verifying a threat or declaring a false alarm. Each task has a label, displayed in the second column.

Table 1. Mission 1- Task sequence for deployed WSN

Mission 1	**Task label**	**Task description**
Input 1	*EVENT u^1*	*UGS1* launches chemical alert
Task 1	*$S4m^1$*	*UGS4* takes measurement
Task 2	*$S5m^1$*	*UGS5* takes measurement
Task 3	*$R1gS2^1$*	*R1* goes to *UGS2*
Task 4	*$R2gA^1$*	*R2* goes to location A
Task 5	*$R1rS2^1$*	*R1* retrieves *UGS2*
Task 6	*$R1lis^1$*	*R1* listens for interrupts
Task 7	*$R1gS1^1$*	*R1* gores to *UGS1*
Task 8	*$R2m^1$*	*R2* takes measurement
Task 9	*$R1dS2^1$*	*R1* deploys *UGS2*
Task 10	*$R1m^1$*	*R1* takes measurement
Task 11	*$S2m^1$*	*S2* takes measurement
output	*y^1*	Mission 1 completed

The input task sequencing matrix F_v corresponding to this mission 1 is

$$F_v^1 = \begin{array}{c|ccccccccccc|} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} & t_{11} \\ x_1^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_2^1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_3^1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_4^1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_5^1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_6^1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ x_7^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ x_8^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ x_9^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}, \qquad F_u^1 = \begin{array}{c|c|} & u1 \\ x_1^1 & 1 \\ x_2^1 & 0 \\ x_3^1 & 0 \\ x_4^1 & 0 \\ x_5^1 & 0 \\ x_6^1 & 0 \\ x_7^1 & 0 \\ x_8^1 & 0 \\ x_9^1 & 0 \end{array}$$

where the 11 columns are labeled in order corresponding to the tasks. Each row of input TSM corresponds to a rule, *e.g*. the second row says

If (task 1 and task 2 have just been completed) then (fire rule 2).

Note that the trigger event *u1* that caused the mission to initiate firing of its tasks is considered as an external input and has its own input matrix F_u. This allows the trigger events to be considered as external inputs to the DEC in the next section.

The output TSM matrix S_v for this example is detailed in[5], and tells which tasks to perform when each rule fires. It is close to an identity matrix, since essentially task *i* is fired by rule *i*.

Though TSM is the matrix considered in[15,16], we have decomposed it into two portions, namely $TSM = S_v \cdot F_v$, with F_v the (input) TSM and S_v the output TSM. The input TSM F_v, and output TSM S_v capture all the temporal precedence relations between the tasks in a mission. Moreover, they map to a *rule base* that corresponds to the rows of the former and the columns of the latter. As we shall see in the next section, this mapping to a rule base is one of the key ideas responsible for our DEC formulation, which allows formal computations for efficient on-line real-time task sequencing and dynamic resource assignment in team networks. Moreover, it was shown in[2,3,4] that the outputs of HTN planners can in fact be directly placed into the format of matrices F_v and S_v.

2.3 Assigning the Resources: Resource Assignment Matrix (RAM)

As just shown, mission commanders indicate intent by prescribing certain task precedences needed to perform a mission with desired goals. As such, missions consist of partially ordered tasks, whose orderings capture the tactically important aspects of a mission. The details of actual task sequencing are left to the DEC to perform during the Operational Phase in real time as events unfold and resources become available. That is, during the Operational Phase the DEC converts the partial ordering provided by mission commanders into a total ordering that directs the actual sequencing of the tasks performed by the networked team in real time. This Operational Phase mechanism will be described in the next section.

Meanwhile, to complete the Programming Phase, resources must be assigned to the tasks. The resources capable of performing the tasks are assigned by field commanders or the war-fighter, who are familiar with the onsite situation. Task capabilities of robotic resources are also available in ARL CIP Agent Registry, a database holding information about the services that registered devices offer in the network community[18]. In this section we detail how the resources capable of performing the tasks are prescribed.

To capture the assignments of the available resources to the mission tasks in a convenient and computable form, define the resource assignment matrix (RAM), which has element *(i,j)* equal to 1 if resource r_j may be used to accomplish task t_i. Note that multiple 1's in a single row *i* indicate that multiple resources are required for task t_i. RAM has been used by[19] and others in manufacturing and elsewhere.

Multiple 1's in column *j* of RAM indicate that resource r_j is needed for multiple tasks. Such *shared resources* are important in the team, as they may be competent or versatile resources that are in high demand. However, shared resources can lead to bottlenecks or catastrophic failures of the team if they are not properly assigned in real time to tasks, or properly *dispatched*. The dispatching of shared resources has been considered under many topics including bottlenecks, deadlocks, and other blocking phenomena[20]. DEC presented in this paper can accommodate the dispatching of shared resources to avoid conflicts and blocking, as discussed in the next section.

RAM can be assigned by field commanders who know which resources can perform which tasks. RAM could also be constructed by software tools that use pricing strategies or payoff matrix ideas, which result in the optimal assignment of resources to tasks given certain prescribed cost functions[21].

As situations change and resources fail or additional resources become available, the resources capable of performing the tasks may change. The DEC framework presented in the next section can accommodate time-varying RAM. Thus, RAM can be modified as resources fail or are added to the team.

To construct a mathematical formulation for a DEC that has guaranteed performance and can work in real-time, we must introduce a rule base. Therefore, factor the RAM matrix into two portions according to

$$RAM = S_v \cdot F_r$$

with S_v the output TSM matrix defined above and dot denoting *or/and* matrix multiply. The input RAM F_r maps the resources $\{r_j\}$ to a rule base $\{x_i\}$ as defined in the previous subsection. Thus, F_r has entry *(i,j)* equal to 1 if resource *j* is required to fire rule *i*. Output matrix S_v maps the rule base $\{x_j\}$ to the tasks $\{t_i\}$. It has entry *(i,j)* of 1 if task *i* is to be stared when rule *j* fires.

Though others have used RAM in analysis[19], we have further decomposed it to map from resources to tasks through a rule base. As seen in the next section, this is one of the keys that makes our DEC useful for real-time control of networked teams.

Networked Team Example. The input RAM for the Mission 1 in Table 1 is

$$F_r^1 = \begin{array}{c|ccccccc} & R1 & R2 & S1 & S2 & S3 & S4 & S5 \\ \hline x_1^1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ x_2^1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_3^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_4^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_5^1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ x_6^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_7^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_8^1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ x_9^1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

where the columns correspond to the available resources and the rows to the rules. Thus, to fire rule 1, one requires sensors *S4* and *S5* to be currently available, etc. That is, rule 1 fires when the event *u1* occurs (as detected by sensor *S1*, see F_u matrix) and if resources *S4* and *S5* are available.

Define likewise an output RAM S_r that maps from the rule base to the resources. This matrix has entry *(i,j)* of 1 if resource *i* is to be released when rule *j* fires. Then a matrix that maps from resources to resources given by

$$G_r = S_r \cdot F_r$$

is the *resource dependency matrix*. G_r defines the so-called resource graph, whose nodes are the resources and whose edges *(j,i)* correspond to entries of 1 in entry *(i,j)* of G_r. This graph is indispensable in studying conflict and deadlock avoidance in systems with shared resources[22,20].

2.4 Programming Multiple Missions

Multiple missions can be programmed into the same networked team of agents and resources. Using DEC, the various missions in a team can be programmed by different mission commanders. Each one does not need to know about other missions running in the network, or about the resources required by the other missions. All missions use the same common pool of networked team resources.

At any time, additional missions can be programmed by other mission commanders, without having to know which missions are already programmed to the resource network. At any time, the resources assigned to the tasks can be changed as resources fail or are added to the network.

As seen in the next section, during the Operational Phase DEC effectively and fairly sequences the tasks of all programmed missions and assigns the required resources on-line in real time as events occur and as resources become available. DEC programs multiple missions into a heterogeneous team of multiple networked resources.

Suppose several missions are prescribed, with Mission *i* having its task ordering given by input TSM F_v^i and its required resources for the tasks given by the input RAM F_r^i. Then the overall TSM and RAM are given by the block matrix compositions

$$F_v = \begin{bmatrix} F_v^1 & 0 & 0 \\ 0 & F_v^2 & 0 \\ 0 & 0 & \ddots \end{bmatrix}, \quad F_r = \begin{bmatrix} F_r^1 \\ F_r^2 \\ \vdots \end{bmatrix}.$$

and similarly for S_v, S_r. Note that the mission task sequences are independent, each using its own tasks, so that F_v is block diagonal. However, all the missions use the same resources available in the networked team, and so have commensurate columns of their resource assignment matrices.

DEC facilitates *mission transferability* between teams by capturing mission information in the TSM, which can easily be moved and programmed into another network.

Networked Team Example. In illustration, consider the same WSN of UGS and mobile robots used in the example above for Mission 1. Suppose the network maintenance technician programs into the same network a Mission 2 that is involved with charging the batteries of the nodes. Such a Mission 2 appears in Table 2. Trigger event *u2* is a low battery event.

Table 2. Mission 2-Task sequence for deployed WSN

Mission 2	**Task label**	**Task description**
input	*EVENT* u^2	*UGS3* batteries are low
Task 1	$S1m^2$	*UGS1* takes measurement
Task 2	$R1g\ S3^2$	*R1* goes to *UGS3*
Task 3	$R1cS3^2$	*R1* charges *UGS3*
Task 4	$S3m^2$	*UGS3* takes measurement
Task 5	$R1dC^2$	*R1* docks the charger
output	y^2	Mission 2 completed

The input TSM and input RAM for this mission are

$$F_v^2 = \begin{array}{c|ccccc} & t_1 & t_2 & t_3 & t_4 & t_5 \\ \hline x_1^2 & 0 & 0 & 0 & 0 & 0 \\ x_2^2 & 1 & 0 & 0 & 0 & 0 \\ x_3^2 & 0 & 1 & 0 & 0 & 0 \\ x_4^2 & 0 & 0 & 1 & 0 & 0 \\ x_5^2 & 0 & 0 & 0 & 1 & 0 \\ x_6^2 & 0 & 0 & 0 & 0 & 1 \end{array}, \qquad F_r^2 = \begin{array}{c|ccccccc} & R1 & R2 & S1 & S2 & S3 & S4 & S5 \\ \hline x_1^2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ x_2^2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_3^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_4^2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ x_5^2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ x_6^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

The overall TSM and RAM for both the missions now in the wireless sensor network are

$$F_v = \begin{bmatrix} F_v^1 & 0 \\ 0 & F_v^2 \end{bmatrix}, \quad F_r = \begin{bmatrix} F_r^1 \\ F_r^2 \end{bmatrix}$$

In similar fashion, additional missions are easily programmed into this WSN.

3. RULE-BASED DISCRETE EVENT CONTROLLER (DEC)

This section describes the DEC software controller that runs in the Operational Phase to sequence the tasks and assign the team resources in real time. This is a decision-making DEC that contains rules to sequence the tasks in each mission, and to assign resources to those tasks. See Fig. 1. DEC has a message-passing architecture that is in conformance with Joint Architecture for Unmanned Ground systems (JAUGS)[10], and is an efficient means to realize the OODA loops (observe, orient, decide, act) of 4D/RCS[11]. DEC is able to coordinate the sequencing of operations between Soldiers and robots efficiently and without conflict, thus contributing to the concept of Safeops.

In the previous section we saw that missions are programmed into the network by specifying task sequencing matrices F_v, S_v and resource assignments matrices F_r, S_r. The TSMs give a partial order for the tasks in each mission, however, the tasks are coupled through the shared resources as captured in RAM. Based on those constructions, it is now desired to construct a rule-based discrete event controller that can be programmed in software and which effectively reacts to external events sensed, sequences the tasks of a networked team, and assigns their available resources in such a way that the missions are accomplished without interference or blocking phenomena. Tasks of priority missions should have priority assignment of requisite resources. How is this to be done?

3.1 DEC State Equation

In terms of the constructions just given we are now in a position to define such a DEC. Define the task vector *v*, resource vector *r*, and rule state vector *x*

$$v = \left[t_1\ t_2 \cdots t_{N_t} \right]^T, \quad r = \left[r_1\ r_2 \cdots r_{N_r} \right]^T, \quad x = \left[x_1\ x_2 \cdots x_{N_x} \right]^T$$

where the set of tasks is $\left\{ t_i; i = 1, N_t \right\}$, the set of resources is $\left\{ r_i; i = 1, N_r \right\}$, and the set of rules is $\left\{ x_i; i = 1, N_x \right\}$. The rule state vector *x* has 1's in positions *i* corresponding to the rules that are currently enabled to fire. Define v_c as the *task completion vector* which contains 1's in positions *i* corresponding to the tasks t_i that have just been accomplished, and r_c as the *resource available vector* containing 1's in positions *i* corresponding to the resources r_i that are currently available. These are the outputs of the networked team passed through messages to the DEC. See Fig. 1. For instance, $v_c = [1\,0\,0\,1\,0 \cdots]^T$ signifies that tasks 1 and 4 have just been performed, while $r_c = [0\,0\,1\,1\,0 \cdots]^T$ signifies that resources 4 and 5 are currently available. Define the *external trigger event vector*

$$u = \left[u_1\ u_2 \cdots u_{nu} \right]^T$$

to contain 1's in positions corresponding to trigger events u_i that have just occurred.

In terms of the TSM and RAM matrices defined above, define the DEC rule base state equation

$$\overline{x} = F_v \cdot \overline{v}_c \ \oplus\ F_r \cdot \overline{r}_c \ \oplus\ F_u \cdot \overline{u} \ \oplus\ F_D \cdot \overline{u}_D \qquad (1)$$

where F_u is an input matrix that specifies which external trigger events are to be used to launch each mission. The input u_D is a *conflict resolution control input* that decides which task to perform in the event that multiple tasks are enabled at a given time. It allows real-time interrupts and priority dispatching for urgent missions, and assigns shared resources in such a way as to avoid blocking phenomena including deadlocks and bottlenecks. See[20].

In this equation, all matrices and vectors are binary, i.e. having entries of either 0 or 1. Dot denotes matrix multiply, and $\oplus$ denotes matrix addition, with all operations carried out in the or/and algebra, where multiplication is replaced by *.and.*, and addition by *.or.*. The overbar denotes negation of all entries of a vector. Operations in the or/and algebra are easily programmed, and a routine that carries out a matrix multiply in the or/and algebra is given in Fig. 3.

The DEC equation contains the required mission task partial orderings and the resources required for each task, and essentially captures the world model in 4D/RCS[11].

```
Matrix Multiply
$C = A \cdot B$
for i= 1,I
  for j= 1,J
    c(i,j)=0
    for k= 1,K
       c(i,j)= c(i,j) .OR. ( a(i,k) .AND. b(k,j) )
    end
   end
end
```

Fig. 3. Matrix multiply in the or/and algebra.

3.2 DEC Output Equations

Based on the rule state vector, the task start equation

$$v_s = S_v \cdot x \qquad (2)$$

computes the *task start vector* v_s, which has 1's in positions *i* corresponding to those tasks that can now be started. The resource release equation

$$r_s = S_r \cdot x \tag{3}$$

computes the *resource release vector* r_s, which has 1's in positions i corresponding to those resources that can now be released as their tasks have ben completed. These output equations are also computed in the or/and algebra.

3.3 DEC as a Feedback Controller

DEC functions as a *feedback controller*, as shown in Fig. 1. It runs as a software tool on a laptop or other computer. It is very easy, for instance, to program DEC in software. Basic code is given in[6]. The operation of DEC is as follows. At each event iteration, all active agents in the team send updates of their tasks just completed and resources currently available over the internet or via WSN. The current task completion vector v_c and the current resource available vector r_c are constructed by taking this information from all active agents in the team. They are considered as the *outputs* of the networked team to the DEC. The DEC then uses state equation (1) to compute the rule state vector x. The resulting entries of 1 in the rule state vector show which rules are enabled to fire, as having all their requisite precursor tasks done and all their required resources available. Thus, the tasks (2) corresponding to these active rules could now be performed. Now among all the tasks that could fire, DEC selects the tasks to actually fire by consulting the mission priorities, or by querying local field commanders via PDA. Finally, command inputs are sent by the DEC telling agents which tasks to start (v_s) and which resources to release and make available (r_s). For autonomous machines, this information is sent as commands to their internal controllers. For human nodes, the information can be sent as decision assistance via a handheld PDA- *e.g*. 'go to point A', 'contact node B and provide certain information', *etc*.

Note that agent nodes need only communicate to DEC when they have a *change* in tasks completion status, or a *change* in resource availability; i.e. when an event occurs. On the other hand, DEC only communicates to those nodes which should next fire tasks or release resources.

3.4 Properties of DEC

The intent of the DEC is that it should provide a mathematically rigorous software tool for implementing a rule-based supervisory controller that sequences the tasks and assigns the resources of a networked team all in real time as events unfold, given at each event iteration the measured network information about which tasks have just completed and which resources are available. This is all governed by the TSM F_v and the RAM F_r, which have been programmed into the DEC software respectively by the mission commanders and the field commanders.

3.4.1 Functionality of the DEC

The first result shows that DEC state equation (1) actually does compute which tasks to start based on rules of the form

IF (all tasks required as immediate precursors to rule i have just been completed)
AND (all resources required by rule i are available) THEN fire rule i

Define the tasks as $\{t_j;\, j = 1, N_t\}$, i.e. the elements of task vector *v*, and the resources as $\{r_j;\, j = 1, N_r\}$, the elements of resource vector *r*. Define T_i as the set of tasks that are immediate precursors to rule *i*, and R_i as the set of resources required to fire rule *i*. The next result verifies the proper functioning of DEC equation (1), while also showing the need for the negation overbars on the vectors in (1).

Theorem 1. Proper Functioning of DEC

The ***i-th*** rule (i.e. *i-th* row) of (1) is equivalent to

$$x_i = \bigcap_{j \in T_i} t_j \cap \bigcap_{j \in R_i} r_j$$

where $\bigcap$ denotes logical *and*. That is, rule state x_i is true (equal to 1) if all task vector elements t_j required for rule *i* are true and all resource vector elements r_j required for rule *i* are true.

Proof: Let $\bigcap$ denote *and* and $\bigcup$ denote *or*. Overbar denotes negation. Define the elements of matrix F_v by f_{ij}^{v} and of F_r by f_{ij}^{r}. By the definition of matrix operations in the or/and algebra, one has

$$\overline{x}_i = \left(\bigcup_{j=1}^{N_T} f_{ij}^v \cap \overline{t}_j \right) \cup \left(\bigcup_{j=1}^{N_R} f_{ij}^r \cap \overline{r}_j \right)$$

Now successive applications of de Morgan's laws yields

$$x_i = \overline{\left(\bigcup_{j=1}^{N_T} f_{ij}^v \cap \overline{t}_j \right) \cup \left(\bigcup_{j=1}^{N_R} f_{ij}^r \cap \overline{r}_j \right)} = \overline{\left(\bigcup_{j=1}^{N_T} f_{ij}^v \cap \overline{t}_j \right)} \cap \overline{\left(\bigcup_{j=1}^{N_R} f_{ij}^r \cap \overline{r}_j \right)}$$

$$x_i = \left(\bigcap_{j=1}^{N_T} \overline{f_{ij}^v \cap \overline{t}_j} \right) \cap \left(\bigcap_{j=1}^{N_R} \overline{f_{ij}^r \cap \overline{r}_j} \right) = \left(\bigcap_{j=1}^{N_T} \overline{f}_{ij}^v \cup t_j \right) \cap \left(\bigcap_{j=1}^{N_R} \overline{f}_{ij}^r \cup r_j \right)$$

But elements f_{ij}^v, f_{ij}^r are equal to zero if task t_j, resp. resource r_j, is *not* needed to fire rule *i*. Then, $\overline{f}_{ij}^v$, $\overline{f}_{ij}^r$ are equal to 1, so that for those elements one has $\overline{f}_{ij}^v \cup t_j = 1$ and $\overline{f}_{ij}^r \cup t_j = 1$ whether the corresponding task or resource element is true or not. On the other hand, elements f_{ij}^v, f_{ij}^r are equal to 1 if task t_j, resp. resource r_j, is needed to fire rule *i*. Then, $\overline{f}_{ij}^v$, $\overline{f}_{ij}^r$ are equal to 0, so that for those elements one has $\overline{f}_{ij}^v \cup t_j = 1$ and $\overline{f}_{ij}^r \cup r_j = 1$ only if the corresponding task or resource element is true. Therefore the last equation is equivalent to

$$x_i = \bigcap_{j \in T_i} t_j \cap \bigcap_{j \in R_i} r_j \qquad \text{QED}$$

3.4.2 Properness of the DEC Rule Base and Fairness of the DEC

The next result shows that the rule base constructed by programming the mission tasks into the TSM and the task resources into RAM produces a rule base of good structure if and only if *each mission* is properly defined. It is shown that improper definition of a single mission assigned to the team can cause improper execution of all missions assigned to the team. Then, the DEC cannot fairly assign the team resources and ensure sequential execution of the tasks in the missions. In extreme cases, improper definition of one mission can tie up resources so that other missions are blocked.

A rule has the form

IF antecedent THEN consequent

where the antecedent consists of task clauses and resource clauses such as

(all tasks required as immediate precursors to rule i are done)
AND (all resources required by rule i are available)

and the consequent consists of a single clause such as

fire rule i.

Problems with rule bases fall into three categories: consistency, completeness, and conciseness [Gursaran 1999].

Consistency of a rule base is compromised by circularity and conflicts.

Lemma 1. The DEC rule base has no circularity or conflicts.

Proof Outline: The rule base is based on clauses such as task completion, and resource availability, and does not contain any of their negations. Therefore it cannot have conflicts. Matrix F_v is block diagonal. Each block defines a strict partial ordering and so is lower block triangular with zero diagonal. Therefore there are no circular chains of rules. QED.

Conciseness is compromised by the presence of rules that logically serve no purpose. These include redundant rules, subsumed rules, and unnecessary IF conditions.

Lemma 2. The overall rule base defined by all the missions is concise if and only if task sequencing matrix of each mission corresponds to a concise rule base for the tasks.

Proof Outline: Matrix F_v is block diagonal, each block of which defines a rule base. Therefore, two rules in different blocks cannot be redundant as they cannot have the same antecedent. No two rules in a single block are redundant by hypothesis. Therefore there are no redundant rules in DEC. The antecedent of a rule in one block cannot contain the antecedent of a rule in another block. By hypothesis, the antecedent of a rule in one block does not contain the antecedent of a rule in the same block. Therefore, there are no subsumed rules. There are no unnecessary IF conditions since, by construction, no clause is the negation of any other. QED.

Completeness refers to knowledge gaps in the rule base. Such gaps make it impossible to achieve the prescribed goals given the triggering events that have occurred. Knowledge gaps include unreachable conclusions, dead end goals, dead end if conditions, and missing rules. By using the block diagonal structure of F_v it is straightforward to prove the following.

Lemma 3. The overall rule base defined by all the missions is complete if and only if task sequencing matrix of each mission corresponds to a complete rule base for the tasks.

These results make it clear that each mission planner has a great responsibility in properly defining his task prior to programming it into the team. Improper definition of one mission among many assigned to a team can cause blocking, thereby tying up resources and making it impossible to complete other missions in the team.

3.5 Shared Resource Dispatching: Blocking, Bottlenecks, and Deadlocks

It has just been shown that if each mission individually is properly defined, the DEC (1) guarantees proper sequencing of tasks in each mission and proper assignment of the required resources. However, the resources of the team are shared by all the missions. Therefore, there may exist bottlenecks or blocking phenomena if the resources are not properly assigned in real time. This is called the shared resource *dispatching* problem. Particularly detrimental is the occurrence of deadlocks, where some tasks cannot gain access to their required resources because those resources are indefinitely held up by other tasks. This indefinitely halts the involved missions and will not allow their completion.

It is shown in[20] that DEC allows easy analysis of potential deadlocks. Then the dispatching or conflict resolution input u_D in (1) (see Fig. 1) may be selected to avoid deadlock situations. In this fashion, the DEC can guarantee proper performance and completion of all missions, as long as each mission is properly defined in the sense shown above in Lemmas 1-3 above.

4. DEC SIMULATION AND IMPLEMENTATION

DEC is easy to implement using computer simulation software. The basic code necessary to implement the DEC shown in Fig. 1 is based on the or/and multiply routine in Fig. 3, and is given in[6]. Messages are passed (using Wifi, internet, WSN, PDA, *etc.*) from team nodes to the C2 computer whenever an event occurs: *i.e.*, any node finishes a task or has a new resource made available. New task events are placed into task completion vector v_c, while new resource events are placed into resource available vector r_c. Then the DEC state equation (1) is evaluated using the software, and the tasks to be started are computed using (2), while resources to be released are computed using (3). If several tasks are enabled, user-specified priority decisions or deadlock considerations are used to select which task to actually fire. Messages are passed back to the team nodes detailing which tasks to perform next v_s and which resources to release r_s, as commands into machine nodes, or as decision aids via PDA to human agents.

4.1 Simulation of Networked Team Example

The DEC was run on the networked team example whose TSM and RAM were shown above. This is a simulation on a digital computer. The resulting event traces are seen in Fig. 4. Chemical attack event u1 occurred at time 8 min, and the low battery event at time 3 min. The progress of the two missions through the team as the resources are assigned and the tasks are performed is clearly seen. In the task traces, 'up' means a task is being performed, while in the resource traces, 'down' means the resource is being used.

In the figure, Mission 1 terminates at 128 min, while Mission 2 terminates at 87 min. In Fig. 5, the priority of the missions in changed, so that Mission 1 is given a higher priority. Thus, when there is a request for the same resource by two tasks, one from each mission, DEC will now assign the Mission 1 task first. This is accomplished in DEC by proper choice of the conflict resolution input u_D in (1) (See also Fig. 1). Details are given in[5]. Now, Mission 1 takes less time and terminates at 62 min.

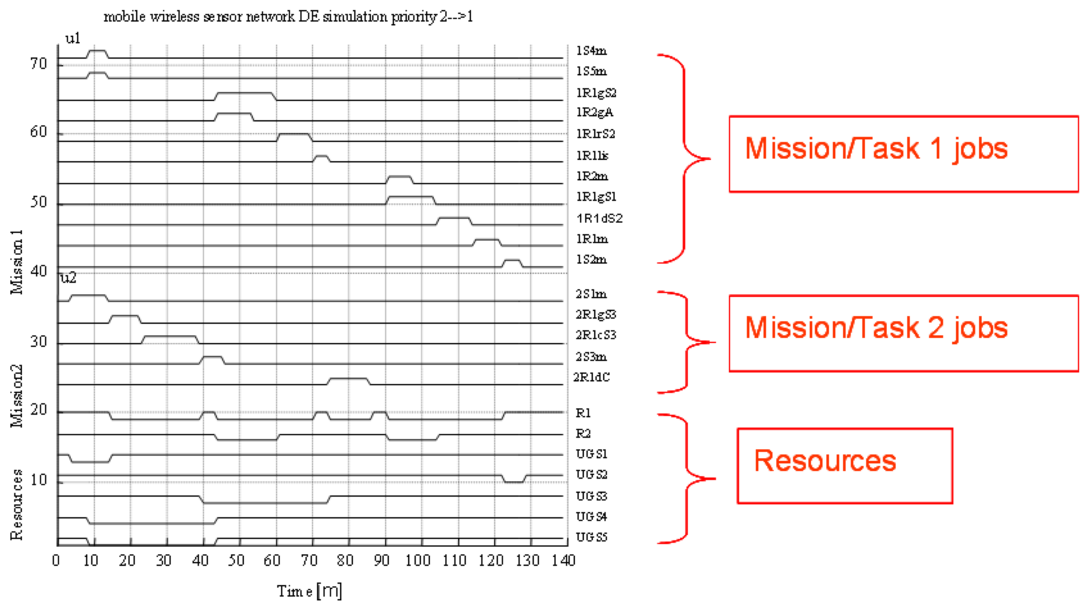


Fig. 4. DEC sequencing mission tasks in the Networked Team Example. Simulation.

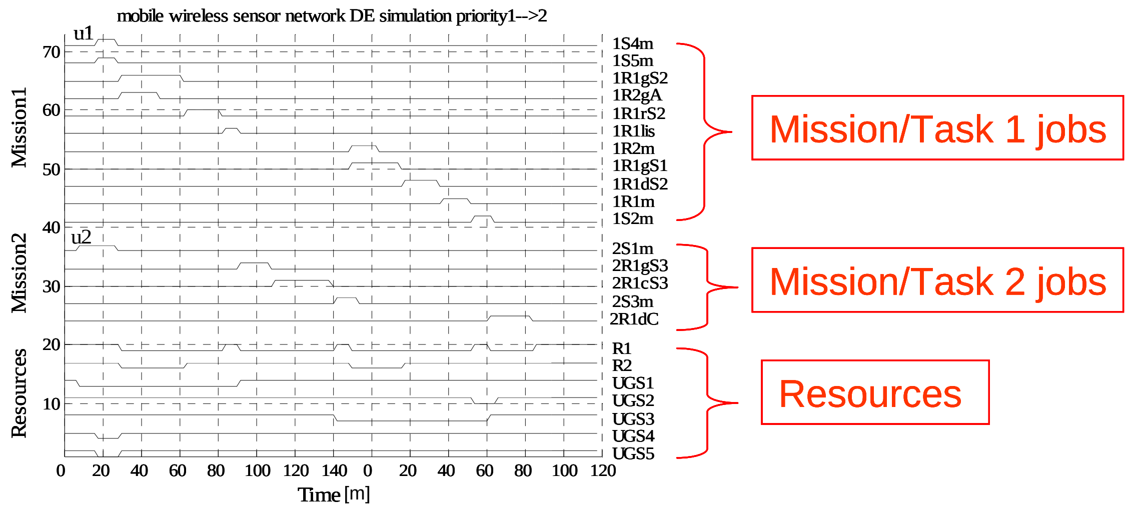


Fig. 5. DEC sequencing mission tasks with increased Mission 1 priority. Simulation.

4.2 Implementation of Networked Team Example on Actual WSN

It is very easy to implement DEC on an actual networked team. In fact, the same code is used for both simulation and implementation. The DEC was implemented on a WSN team of mobile robots and UGS at the UTA Automation & Robotics Institute. Details of the hardware are given in[5]. A VR 3D user interface depicted the motions of the robots during the mission execution. The panoramic view during the mission execution is shown in Fig. 6. The actual event traces observed during the experimental implementation are shown in Fig. 7. They bear a close resemblance to the simulated event traces above.

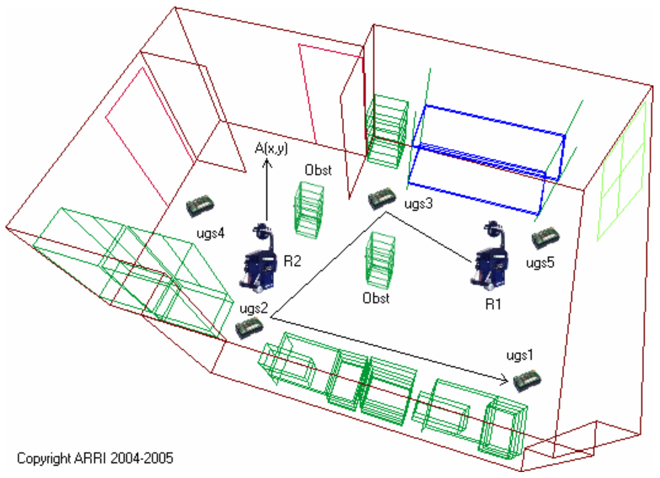


Fig. 6. DEC VR interface panoramic view of the configuration of the mobile WSN during real-world experiments.

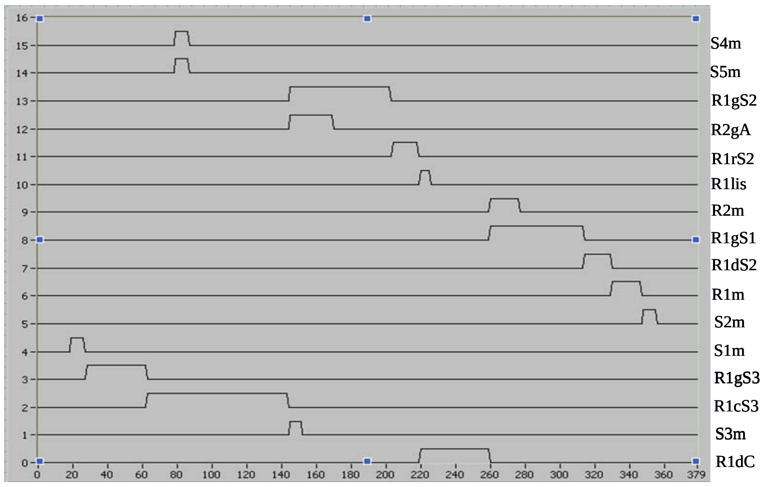


Fig. 7. Task event trace of the WSN- Experimental results.

ACKNOWLEDGEMENTS

This work is supported in part by ARO grant ARO W91NF-05-1-0314 and the Army National Automotive Center.

REFERENCES

[1] F.L. Lewis, A. Gurel, S. Bogdan, A. Doganalp, O. Pastravanu, “Analysis of deadlock and circular waits using a matrix model for flexible manufacturing systems,” Automatica, vol. 34, no. 9, pp. 1083-1100, 1998.

[2] Harris, B., Cook, D. J., and Lewis, F. L., “Automatically generating plans for manufacturing,” Journal of Intelligent Systems 10, 279-319 (2000).

[3] Harris, B., Cook, D. J., and Lewis, F. L., “Combining representations from manufacturing, machine planning, and manufacturing resource planning (MRP),” Proc. AAAI Workshop on Representational Issues for Real-World Planning Systems (2000).

[4] Harris, B., Lewis, F. L., and Cook, D. J., “Machine planning for manufacturing: dynamic resource allocation and on-line supervisory control,” Journal of Intelligent Manufacturing 9(5), 413-430 (1998).

[5] Giordano, V., Ballal, P., Lewis, F. L., Turchiano, B., and Zhang, J. B., “Supervisory control of mobile sensor networks: math formulation, simulation, and implementation,” IEEE Trans. Syst., Man, Cybern. B 36(4), 806-819 (2006).

[6] Tacconi, D. A., and Lewis, F. L., “A new matrix model for discrete event systems: application to simulation," IEEE Control Syst. Mag. 17(5), 62-71 (1997).

[7] Bogdan, S., Lewis, F. L., Kovacic, Z., and Mireles Jr., J., Manufacturing systems control design: a matrix based approach (advances in industrial control), Springer-Verlag, London (2006).

[8] Gursaran, G. S., Kanungo, S., and Sinha, A. K., “Rule-base content verification using a digraph-based modelling approach,” Artificial Intelligence in Engineering 13(3), 321-336 (1999).

[9] Guida, G., and Mauri, G., “Evaluating performance and quality of knowledge-based system: foundation and methodology,” IEEE Trans. Knowledge Data Eng. 5(2), 204-224 (1993).

[10] Thibadoux, S. A., “Robotic acquisition programs- technical and performance challenges,” in Unmanned Ground Vehicle Technology IV, Proc. SPIE 4715, Orlando, 128-138 (2002).

[11] Albus, J. S., “Intelligent systems design,” Plenary Talk, Army Science Conf., Orlando (2008).

[12] Bornstein, J. A., “Army ground robotics research program,” in Unmanned Ground Vehicle Technology IV, Proc. SPIE 4715, Orlando, 118-127 (2002).

[13] Shah, H. K., Bahl, V., Martin, J., Flan, N. S., and Moore, K. L., “Intelligent behavior generator for autonomous mobile robots using planning-based AI decision-making and supervisory control logic,” in Unmanned Ground Vehicle Technology IV, Proc. SPIE 4715, Orlando, 161-177 (2002).

[14] Smuda, W., Muench, P., Gerhart, G., and Moore, K. L., “Autonomy and manual operation in a small robotic system for under-vehicle inspections at security checkpoints,” in Unmanned Ground Vehicle Technology IV, Proc. SPIE 4715, Orlando, 1-12 (2002).

[15] Steward, D. V., “The design structure system: a method for managing the design of complex systems," IEEE Trans. Eng. Manage. 28(3), 71-74 (1981).

[16] Warfield, J. N., “Binary matrices in system modeling,” IEEE Trans. Syst., Man, Cybern. 3(5), 441-449 (1973).

[17] Wolter, J. D., Chakrabarty, S., and Tsao, J., “Methods of knowledge representation for assembly planning,” Proc. 18th Annual NSF Conference on Design and Manufacturing Systems Research, Atlanta, GA, USA, 463-469 (1992).

[18] Young, S. H., and Nguyen, H. M., “System design for robot agent team,” in Unmanned Ground Vehicle Technology IV, Proc. SPIE 4715, Orlando, 31-42 (2002).

[19] Kusiak, A., and Ahn, J., “Intelligent scheduling of automated machining systems,” Computer-Integrated Manufacturing Systems 5(1), 3-14(1992), Reprinted in A. Kusiak (Ed.), Intelligent Design and Manufacturing, John Wiley, New York , NY, 421-447 (1992).

[20] Gurel, A., Bogdan, S., and Lewis, F. L., “Matrix approach to deadlock-free dispatching in multi-class finite buffer flowlines,” IEEE Trans. Automat. Contr. 45(11), 2086-2090 (2000).

[21] Huang, H.-H., Gmytrasiewicz, P. J., and Lewis, F. L., “Combining matrix formulation and agent-oriented techniques for conflict resolution in manufacturing systems,” Proc. ACME, Chicago, 66-79 (1996).

[22] Wysk, R. A., Yang, N. S., and Joshi, S., “Detection of deadlocks in flexible manufacturing cells,” IEEE Trans. Robot. Automat. 7(6), 853-859 (1991).