Network-centric IED detection planning

Kyle Usbeck* and Jeffrey Cleveland

BBN Technologies,
10 Moulton St.,
Cambridge, MA 02138, USA
E-mail: kusbeck@bbn.com
E-mail: jclevela@bbn.com
*Corresponding author

William Regli

Drexel University,
3141 Chestnut St.,
Philadelphia, PA 19104, USA
E-mail: regli@cs.drexel.edu

Abstract: As methods for detecting improvised explosive devices (IEDs) continue to diversify, it becomes increasingly important to establish a framework for coordinating distributed IED monitoring resources to best protect a designated area. The purpose of this paper is to establish the beginnings of such a framework in a distributed plan execution context. The first contribution of this paper is defining an automated planning domain for distributed IED detection. In doing so, we investigate approaches for coordinating distributed plan execution resources. Whereas many existing multi-agent system (MAS) frameworks abstract network information from agent decision-making processes, we instead propose that MAS frameworks consider network properties to improve effectiveness. The second contribution of the paper is the description of several types of network-aware planning, execution, and monitoring agents and a comparison of their performance and effectiveness in an IED monitoring scenario. The results of this research indicate that network-awareness improves distributed plan execution (e.g., IED monitoring). Furthermore, we find that an agent's benefit from reasoning about network properties is directly affected by the amount of dynamism in the network.

Keywords: IED detection; distributed systems; automated planning; network-centric systems; network-aware agents.

Reference to this paper should be made as follows: Usbeck, K., Cleveland, J. and Regli, W. (2012) 'Network-centric IED detection planning', *Int. J. Intelligent Defence Support Systems*, Vol. 5, No. 1, pp.44–74.

Biographical notes: Kyle Usbeck works at Raytheon BBN Technologies as a Software Engineer and Scientist in the distributed systems group. There, he has contributed to projects involving quality of service (QoS) over airborne networks, spatial computing, information management systems, and electro-mechanical system design modification. Prior to that, he worked as a Lead Software Engineer at a start-up that specialises in situation-awareness software for law enforcement, emergency response, and first-responders. He received his Masters of Science and Bachelors of Science in Computer Science from Drexel University.

Jefffrey Cleveland works at Raytheon BBN Technologies as a Staff Scientist in the distributed systems group. There, he has contributed to projects related to modelling military scenarios, fault tolerant information management systems, intelligent information life-cycle management, trusted computing techniques, and electro-mechanical system design modification. He received his MS and BS in Computer Science from the University of Massachusetts Amherst.

William Regli is a Professor in the Department of Computer Science in the College of Engineering at Drexel University with joint appointments in the Department of Mechanical Engineering and Mechanics and Electrical and Computer Engineering. He is also Executive Director of The AJ Drexel Applied Communications and Information Networking (ACIN) Institute, a university-wide multidisciplinary research initiative of Drexel University. He received his PhD in Computer Science from the University of Maryland at College Park and BS in Mathematics and Computer Science from Saint Joseph's University. He has previously been a Visiting Researcher at Lockheed Martin's Advanced Technology Laboratories, Research Scientist at AT&T Laboratories' Internet Platforms Technology Organisation, member of the Research Faculty at Carnegie Mellon University and a National Research Council (NRC) Post-Doctoral Research Associate and Computer Scientist at the National Institute of Standards and Technology (NIST).

1 Introduction

As improvised explosive device (IED) detection techniques continue to emerge and diversify, it becomes increasingly important to coordinate the efforts of multiple, heterogeneous detection resources, e.g., unmanned aerial vehicles (UAVs), unmanned ground vehicles (UGVs) and trained personnel. Such coordination can potentially lower the costs (e.g., time, manpower and fuel) and increase the effectiveness of distributed IED detection activities. Multi-agent planning systems are commonly used to coordinate heterogeneous resources such as those outlined here. The purpose of this article is to extend such planning systems with network-aware mechanisms and then evaluate the performance and effectiveness of network-centric approaches to automated planning, plan execution, and plan monitoring in the distributed IED detection scenarios.

The contributions of this article are as follows:

- the definition of an automated planning domain for distributed IED detection
- a description of network-aware planning, execution, and monitoring agents and a comparison of their performance and effectiveness in a distributed IED monitoring scenario.

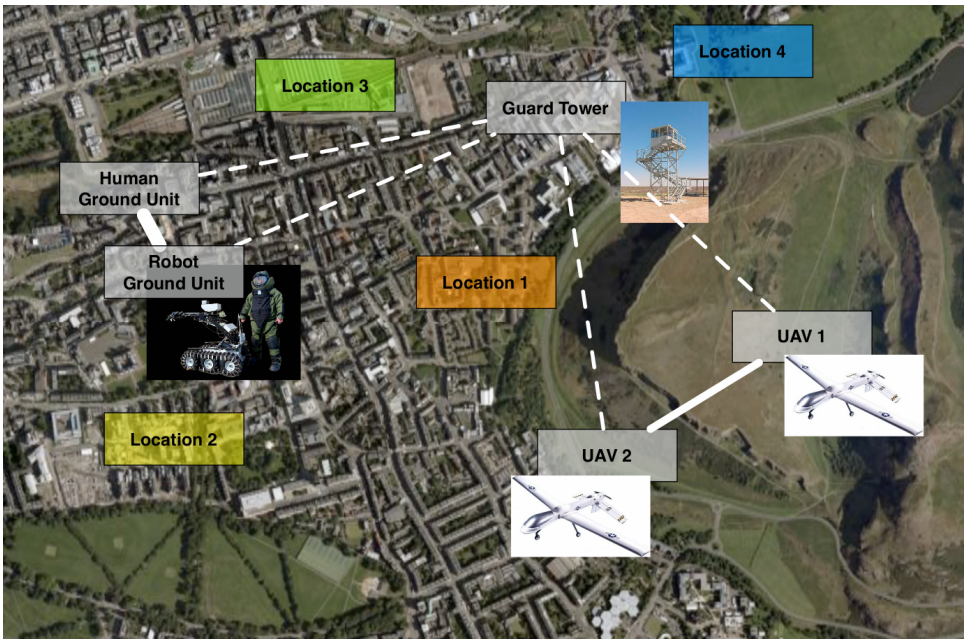
The article has the following organisation: The remainder of this section presents more detail on the IED detection scenario. Section 2 establishes necessary background information. Section 3 first describes our network extensions to the automated planning domain, then presents descriptions of network-aware planning, execution, and monitoring agents. The experiments that we use to determine the performance and effectiveness of each agent are explained in Section 4 and the empirical results are in Section 5. Finally, we present related work and conclusions in Section 6 and 7, respectively.

1.1 Distributed IED Detection Scenario

The distributed IED detection scenario, depicted in Figure 1, consists of heterogeneous actors monitoring key locations for IEDs. Coloured rectangles (Locations 1, 2, 3 and 4) represent key physical locations that require monitoring (e.g., busy intersections, marketplaces, etc.). The white rectangles represent locations of actors capable of carrying out various IED detection techniques. The white lines connecting actors represent the current network topology in the scenario.

The goal is to maximise the effectiveness of IED detection while minimising the overall cost of evaluation. Both the effectiveness and cost are affected by the monitoring techniques selected (e.g., visual change detection, manual searching), the entities that perform the techniques (e.g., UAV, humans, robots), the schedule in which the techniques are executed, and the resources utilised during IED detection (e.g., camera type and resolution, robotic detection accuracy).

Figure 1 The IED detection scenario (see online version for colours)



Note: Several locations are monitored for IEDs using manual searching or visual change detection.

Actors in the distributed IED scenario generally include units such as UAVs, UGVs, trained personnel, stationary sensor nodes, checkpoints, and guard towers. The following are several ways in which actors may vary:

Mobility	Mobile units are capable of travelling to remote locations to perform IED detection tasks with travel time determined by locomotion methods, e.g., an airborne asset is likely to move faster than personnel on foot. Stationary units can only monitor their current area.
Sensing	Sensing capabilities can vary with sensor type, range, and resolution. Likely sensor types include cameras or other visual sensors, chemical sensors capable of detecting explosive components, radar and LIDAR.
Communication	The strength of a communication link will depend on an actors position and its communication hardware. Further, different actors have different communication requirements (e.g., trained personnel may only need intermittent network connectivity to receive plans and report results) and do not need to maintain persistent network links throughout the scenario.
Autonomy	Some units, e.g., UGVs, may require a nearby operator.

These properties can be seen as key domain-specific dependencies for the effectiveness of a given IED detection techniques, e.g., visual change detection, chemical sensor readings, object recognition, manual inspection, and semantic web-based analysis.

2 Background

2.1 Network-centric systems

We adapt the ideas from Peysakhov et al. (2004) to incorporate network-awareness into the agents involved in creating plans, executing plans, and monitoring plan execution. Particularly, we use the concept of the *overlay network* as a formulation of the connections between agents, and generalise their claim that sensing and reasoning about the underlying network can improve the effectiveness of distributed problems, such as coordinated IED detection. We do so by categorising agents according to their goals and analysing their performance (i.e., the utilisation of resources during system execution) and effectiveness (i.e., how well the system accomplishes its goals) under different types of network dynamism (Basili et al., 1994).

We define network-centric systems (see Definition 1) based on Peysakhov's evaluation of network-awareness.

Definition 1: A network-centric system is a distributed system where effectiveness is dependent on the quality of the underlying network communication links.

In other words, a system is network-centric if its effectiveness changes as a result of network fluctuations.

We propose that the effectiveness of actors performing distributed IED detection depend on network quality, and thus are a network-centric system. Network performance is vital for coordinating actions among the many actors in the distributed IED detection scenario and for reporting detected IEDs. Furthermore, many IED detection techniques rely on vast computational power or extensive data stores which likely exceed the resources available on tactical devices such as UAVs and UGVs, and requiring these platforms maintain connectivity to a remote data centre. For instance, visual change detection requires the comparison of current images to archived images of the same geographic area. The limited storage capabilities of actors such as UAVs and UGVs would likely be exhausted by the image archives required to perform visual change

detection for their effective operational area. Similarly, change detection is computationally intensive, and not necessarily suited to the embedded platforms upon which these devices commonly run. Other similar examples include automated object detection which often requires access to large object databases and is computationally expensive, as well as semantic web-based analysis which require massive data stores and computational power.

For these reasons, we see the distribute IED detection network as a network-centric system.

2.2 Automated planning

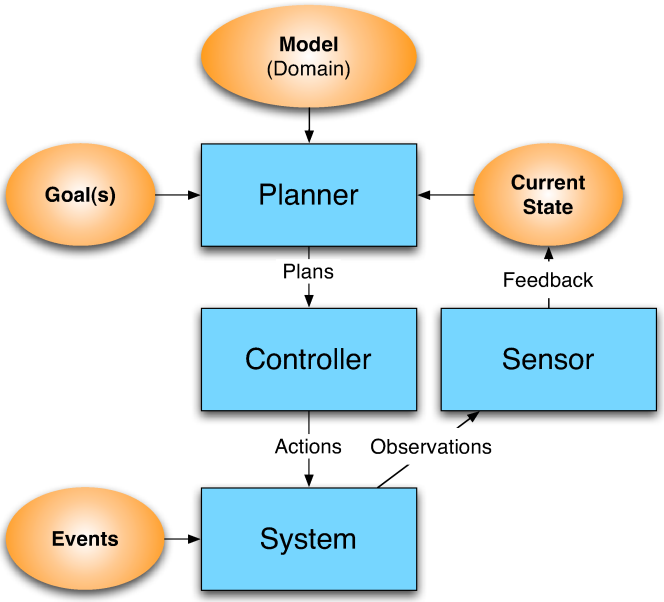
Collaborative IED detection is a difficult problem, largely due to the huge number of variables required to coordinate distributed resources to best monitor some set of geographical areas. *Automated planning* is a branch of artificial intelligence that is designed to determine the best allocation of resources and sequence of actions to accomplish a goal. We draw heavily upon automated planning for modelling the IED detection scenario and determining which resources (e.g., UAVs, human-robot teams, etc.) should perform which actions (e.g., visual change detection, chemical detection, etc.) at what geographical locations.

The remainder of this section discusses:

1 the role of agents in the planning and execution process

2 the classical planning domain model upon which we build our extensions.

Figure 2 The interactions between agents in a planning architecture (see online version for colours)



2.2.1 Agent roles

Tate (1993) discusses the roles of agents in the planning process and defines three key agent roles: *task assignment*, *planning*, and *execution*. In this notion, the planning agent is responsible for solving a static planning problem and passing the plan to the execution agent. The execution agent interacts with the real system, and in some situations, can react to some action execution failures. The task assignment agent communicates with the planning and execution agents to trigger plan creation and execution respectively.

The agent interactions in Tate (1993) form a control theoretic feedback loop. A feedback loop contains a *controller* which initially accepts a plan and then gives input to some *system*, or plant. A *sensor* component determines the current state of the system to help guide the controller in the remaining execution steps. In this terminology, the controller functions as the task assignment and execution agent – it should be noted that these operations can be separated. Figure 2 combines Tate's agent roles with the feedback loop of agents in planning systems.

2.2.2 Classical planning model

We adopt the classical planning notation from Nau et al. (2004) where the planning domain, Σ, is a state transition system such that $\Sigma = (S, A, E, \gamma)$ where S is the set of states, A is the set of actions, E is the set of events, and $\gamma = S \times (A \cup E) \rightarrow 2^S$. Each action, a, is defined by its requirements for execution, precond(a), and its positive and negative post-conditions of execution, effects$^+$(a) and effects$^-$(a).

The planning problem, $\mathcal{P}$, can in turn be expressed as the triple (Σ, s_0, S_g) where s_0 is the initial state and S_g is a set of goal states. Table 1 summarises the notation of the classical planning model for convenience. See Nau et al. (2004) for more details regarding classical planning.

Table 1 Summary of the classical planning model

$\mathcal{P}$	Is the planning problem
Σ	Is the planning domain
s_0	Is the initial state
S_g	Is the set of goal states
S	Is the set of states in Σ
A	Is the set of actions, $\{a_0, a_1, \ldots, a_{\|A\|}\}$ in Σ where each a = (precond(a), effects$^-$(a), effect$^+$(a))
E	Is the set of system events in Σ
γ	Is the state transition function
precond(a)	Returns the preconditions of action, a
effects$^+$(a)	Returns the positive effects of action, a
effects$^-$(a)	Returns the negative effects of action, a

2.3 Plan monitoring

Plan execution monitoring has been studied extensively by control theorists as the *fault detection and isolation* (FDI) problem (Pettersson, 2005). The functional concepts of monitoring systems include:

- *fault detection* – determining that an anomaly has occurred in the behaviour of the monitored system
- *fault isolation* – the classification of the fault (a.k.a. fault diagnosis)
- *fault identification* – determining the magnitude of the fault.

Chiang et al. (2001) list three classifications of execution monitoring. The following sections explain the analytical, data-driven, and knowledge-based approaches in detail.

In the *analytical approach*, the presence of a fault is derived from the difference between two analytically generated quantities. The difference is called *residual* and the algorithm that processes the measurable inputs and outputs to a system is called *residual generation*. The analytical approach utilises some decision making algorithm, $d(r)$ where r is a residual, based on residual generation, $r(s)$ where s is a signal from a system. The process relies on the concept of analytical redundancy and is influenced by the residual generation technique.

Data-driven approaches do not rely on mathematical models, but instead they are directly derived from sensor data. For example, a fault might be detected if sensor data exceeds some range of deviation from the mean of the previously collected data. These approaches are classified by the number of variables included in their monitoring. Single-variable approaches are labelled univariate statistical monitoring and all other approaches are labelled multivariate statistical monitoring.

Knowledge-based approaches to execution monitoring are designed to simulate human problem-solving. They can be model-free, model-based, or some hybrid of other methods. To offset the high cost of human behaviour simulation, knowledge-based monitoring systems often perform fault isolation in addition to fault detection.

3 Approach

This section explains our approach toward creating and evaluating network-aware agents. First, we discuss the sources of uncertainty in the IED detection domain that necessitate this work. Next, we explain our extensions to the planning domain for incorporating the notion of distributed services into a formal planning problem. Then, we explain our approach to creating network-aware planning, execution, and monitoring agents.

3.1 Uncertainty in the IED detection scenario

In our coordinated IED detection scenario, resources collaborate to find and report IEDs that may be present in their operating environment. The resources and areas of importance are fixed, however there remain a large number of variables that are unknown *a priori*. These variables represent the *uncertainty* in the scenario and this uncertainty is the primary difficulty in finding a solution.

In the IED detection scenario, uncertainty exists in several forms. First, the environment can be changed by a variety of factors, not all of which can be realistically modelled. These deviations in the environment can cause problems during execution that may have not occurred otherwise. For example, a large cloud could cast a shadow on an area during a photograph, making visual change detection on that area less-reliable. This source of uncertainty is usually called *domain dynamism* and we account for this by using reactive/proactive execution agents as well as monitoring the progress of plan execution for anomalies. Planning occurs on an as-needed basis and the severity of detected anomalies triggers plan repair (i.e., an update to the plan) or replanning (i.e., generating a new plan).

Next, our distributed agents have limited visibility; the source of uncertainty that arises from lack of visibility is called *partial observability*. We utilise communication between devices to minimise the effects of partial observability. Further, reasoning about how and when to communicate between devices lessens the communication overhead.

Finally, our goal is to find the 'best' solution, rather than 'any' solution to the coordinated IED detection planning. That is, not only do we want a solution that will monitor all locations for an IED, but we also want a solution that can be executed quickly (i.e., in less time), efficiently (i.e., using fewer finite resources e.g., gasoline), and effectively (i.e., to a sufficient degree of reliability). Finding 'any' solution, or *satisfying* the problem, is significantly simpler that finding the 'best' solution, or *optimising* the solution. Difficulty in optimisation arises from the high-dimensionality of the problem-space. We use domain-independent, domain-dependent, and network-centric heuristics (listed in Section 3.3.1 and Section 3.3.3) to better explore the dimensionality of the problem-space and to determine which plans are likely to be the most effective and efficient.

3.2 Planning domain extensions for network-awareness

This section establishes a formal problem statement which builds on the planning notation established in Section 2.2.2. The following extensions are designed to be added to any traditional planning problem to represent distributed services and the network upon which they operate:

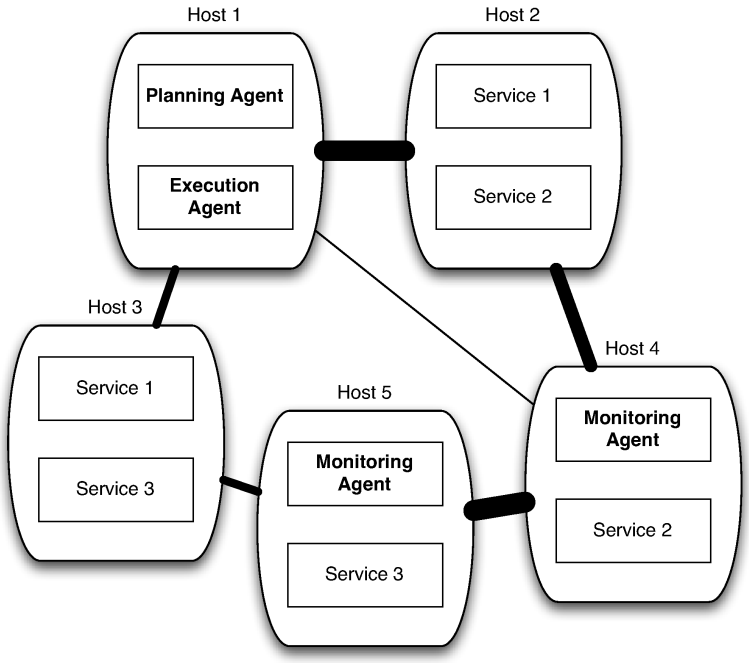
H	The network hosts $H = \{h_0, h_1, \ldots, h_{\|H\|}\}$
ω_H	The host link weighting, ω_H: $H \times H \rightarrow (0, 1]$. The value represents the quality of the link, 0 being the lowest and 1 the highest. This link is only conceptual and the actual data route may go between other hosts or routers
offers(a)	Returns the precondition(s) stating that a host offers the service represented by action, a
host(a)	Returns the single host $h \in H$ on which action a can be run, or $\emptyset$ if the action is ungrounded
resources(a)	Returns the set of resources (parameters) of action a, or {} if the action is ungrounded

Figure 3 shows the conceptual diagram of the formal problem statement. Hosts, $h \in H$ are connected via a network, with links from host h_i to h_j. Each link also has a relative quality, ω_H, illustrated as the thickness of the connecting lines. The value of ω_H can vary based on the domain. For example, if timeliness is critical, ω_H should be some function of

network latency and bandwidth between two hosts. Furthermore, our host link weighting uses the notion of an *overlay network* such that two hosts with a non-zero ω_H may not necessarily be capable of communicating directly, but rather that they can communicate through some means (e.g., through another host in H or through some communication infrastructure such as cellular or satellite networks).

It is important to note, however, that ω_H is not restricted by transitivity or the triangle inequality. That is, a link from h_i to h_j could have a higher or lower ω_H than $\sum[\omega_H]\{h_i \to h_x, h_x \to h_j\}$ for any h_x.

Figure 3 The conceptual diagram of the formal problem statement



Notes: Rounded rectangles represent hosts $h \in H$, which are capable of executing a set of services. Planning, execution, and monitoring agents reside on hosts. Lines between hosts represent communication links between the hosts and the thickness of the line is a representation of link quality, ω_H.

Figure 4 shows the role of agents in the formal problem statement. There are three types of agents: planning, execution, and monitoring. Each agent runs on a host and the agents communicate with each other over the network.

Also, Figure 4 shows the data flow between agents in the formal problem statement. The planning agent is given the tuple $IP = \langle \Sigma, s_0, S_g, H, \omega_H \rangle$ and constructs a plan $p_I \in P$. The execution agent accepts $I_E = \langle p_I, H, \omega_H \rangle$, and invokes the totally ordered service calls described by the actions in p_I. The plans can be augmented with temporal constraints to better describe the sequence of service invocations.

Figure 5 shows the sequence diagram for the plan execution process. Plans are created by the planning agent and passed to the execution agent. The execution agent causes changes to a system, which can be sensed by monitoring agents. When a

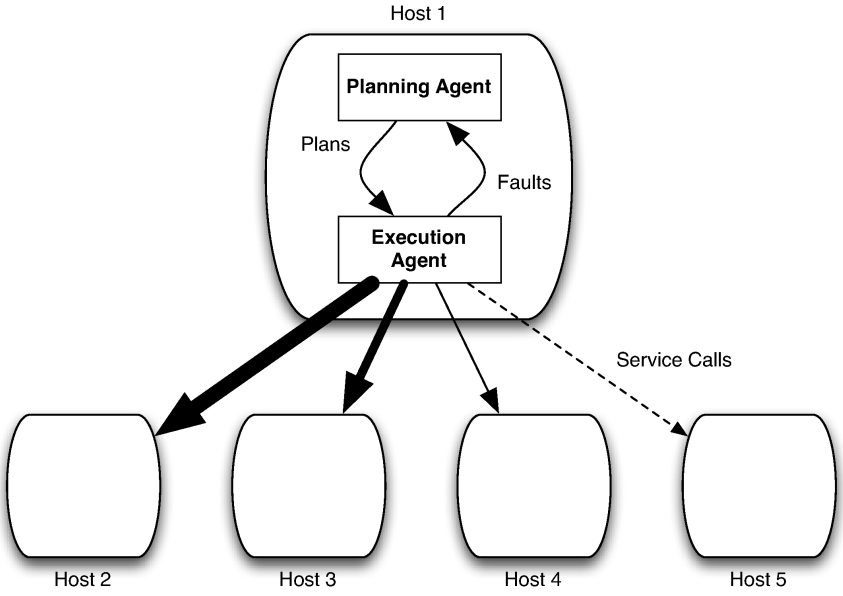
monitoring agent detects a fault, it isolates and identifies the fault (see Section 2.3). Based on the magnitude of the fault and the capabilities of the agents, the monitoring agent reports the fault to the execution agent (for plan repair) or planning agent (for replanning).

The problem is to find and execute $p_I \in P$ where $p_I = \{a_0, a_1, ..., a_{|p_I|}\}$ and the execution of p_I yields the best domain-dependent and network-centric evaluations. Using this goal, we can now formally define *network-awareness* as Definition 2.

Definition 2: An agent exhibits *network-awareness* if changes to ω_H cause the agent's output to change while all other inputs remain constant.

In other words, agents are *network-aware* if changes to *only* the networking conditions (ω_H) cause the agent to act differently. We use this definition to describe our network-aware agents in Sections 3.3, 3.4, and 3.5.

Figure 4 The data flow and role of agents in the formal problem statement



Notes: There are three types of agents: planning, execution, and monitoring. Each agent runs on a host and the agents communicate with each other over the network. The planning agent is responsible for creating plans and passing them to the execution agent. The execution agent invokes services on the hosts according to the plan. The monitoring agents report execution faults to the planning and execution agent as needed. Each network link has a relative quality, ω_H, illustrated as the thickness of the lines connecting hosts. Service-specific, inter-host communications are not illustrated in this diagram.

3.3 Network-aware planning

The planning agent is responsible for producing a plan, the goal of which is to give advice to the execution agent(s). The main implementation differences between the planning agents stem from the technique used to guide the planner's search strategy.

Following the notation introduced in Section 3.2, the purpose of a planning agent is to accept the tuple I_P as input and produce a plan in the form of an ordered service composition p_I.

The remainder of this section discusses three planning agents:

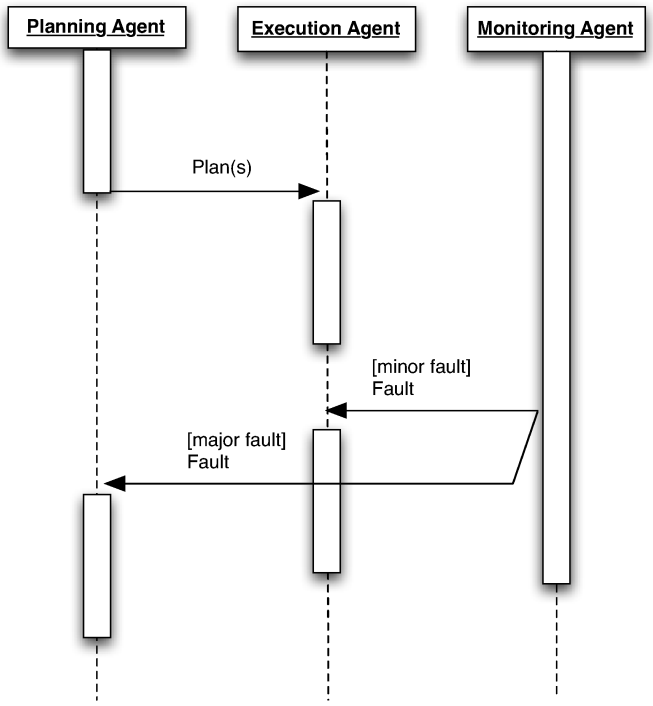
1 the domain-independent planning agent (or I-Plan planning agent)

2 the random planning agent

3 the guided planning agent.

Of these three, the guided planning agent exhibits network-awareness as defined in Definition 2.

Figure 5 The sequence diagram for the plan execution process



Notes: The planning agent is responsible for creating plans and passing them to the execution agent. The execution agent invokes services on the hosts according to the plan. The monitoring agents report execution faults to the planning and execution agent as needed.

3.3.1 Domain-independent planning agent

All planning agents presented in this paper were implemented using Tate et al.'s plan-space HTN planner, I-Plan (Currie and Tate, 1991). The domain-independent planning agent is based on I-Plan's default search strategy, which uses a combination of exploration and optimisation to return different plans. As the planner traverses the search space, it switches between a depth-first exploration strategy and an A* optimisation

strategy using the number of activities in the partial plan as its admissible heuristic. The planner starts by traversing the space in a depth-first manner and when it encounters an alternative whose constraints cannot be satisfied, it backtracks using the A* search and that alternative is removed.

The domain-independent planning agent utilises domain-independent heuristics to guide the planner's search strategy. Included in these heuristics are the following:

- the number of steps the planner took to generate a plan
- the number of alternatives the planner uncovered along its way
- the number of options below the revealed alternatives
- the number of alternatives left unexplored
- the longest path along temporal ordering constraints
- the number of duplicate plans found before returning a new plan.

3.3.2 Random planning agent

The random planning agent conducts a depth-first search, selecting randomly among branch-point alternatives in I-Plan's plan-space search. Algorithm 1 shows the process of the random planning agent.

Algorithm 1 ConstructRandomPlan ($\mathcal{P}$)

Require: $\mathcal{P}$ is the planning problem for which this algorithm constructs a plan to solve.

Ensure: s_0 is the initial state of $\mathcal{P}$, toVisit is a deque of branch-point alternatives in the plan-space, visited is a list of alternatives already traversed, randomise(z) is a function that returns the ordered set z with all its original elements in a random order, and solution(x) is a Boolean function that returns true if x meets solution criteria set by the planner.

```
1:  toVisit.push(s0)
2:  while ¬ toVisit.empty() ^ ¬ solution(toVisit.peek()) do
3:      v ← toVisit.pop()
4:      if v ∉ visited then
5:          visited.add(v)
6:          r ← randomise(v.children())
7:          toVisit.push(r)
8:      end if
9:  end while
10: return toVisit.peek()
```

3.3.3 Plan evaluation guided planning agent

The guided planning agent defines domain-dependent plan evaluators for criteria that are important to the end-user. In this case, we create evaluators for network performance as well as domain-dependent criteria. The guided algorithm seeks to exploit the trade-offs between the defined evaluation criteria by finding qualitatively-different plans.

The guided planning agent is the only planner in the experiments that is network-aware. It utilises network-based plan evaluations and generates qualitatively-different plans over those criteria.

Current work on generating qualitatively-different plans discusses two high-level techniques: domain-independent, and domain-dependent. We use plan evaluation criteria to represent both techniques. The reasoning for using a single mechanism for both techniques is that plan evaluators are sufficiently capable of recognising domain-independent as well as domain-dependent information about a plan.

Our method for biasing the planner's search strategy based on plan evaluations is to maintain a set of priority queues, $\mathcal{L} = \{Q_0, Q_1, ..., Q_{|\mathcal{L}|}\}$ – one for each plan evaluator. Every time a new partial-plan/backtrack-point is generated, its viability is assessed and it is inserted into each priority queue according to the partial plan evaluation of the priority queue's plan evaluator. Psuedocode is found in Algorithm 2.

Algorithm 2 HandlePartialPlan(*p*)

Require: *p* is the partial plan accepted as input. $\mathcal{E}$ is the set of plan evaluators. $\mathcal{L}$ is a list of priority queues containing plan evaluations.

Ensure: evalPartialPlan(*evaluator*, *p*) is a function that evaluates partial plan, *p*, using the partial plan evaluator, *evaluator*. insert(*Q*, *e*, *p*) is a function that inserts a partial plan, *p*, into the priority queue, *Q*, according to the evaluation, *e*.

1: **for all** *evaluator* $\in \mathcal{E}$ **do**
2: $Q \leftarrow \mathcal{L}$ [*evaluator*]
3: $e \leftarrow$ evalPartialPlan(*evaluator*, *p*)
4: insert(*Q*, *e*, *p*)
5: **end for**

In addition to the domain-independent criteria that is listed in Section 3.3.1, the guided planning agent also uses domain-dependent and network-centric heuristics. Included in the domain-dependent heuristics are the following:

- estimated IED detection accuracy (measured as a percentage of certainty that an IED would be discovered if present)
- plan execution time (measured as the time required to complete all plan actions).

Our network-centric heuristics include the following:

- network bandwidth usage (measured in Mbps)
- number of network hops (measured as the total number of links that are required along the data-path).

The planner uses the greedy strategy discussed in Srivastava et al. (2006) to generate multiple plans. The planner iterates over the plan evaluators for each plan it generates. Thus, every new plan represents an alternative from the head of a different evaluator queue.

3.4 Network-aware execution

The execution agent is responsible for executing a set of actions on a system. Different types of execution agents are defined formally in this section.

Following the notation introduced in Section 3.2, the purpose of an execution agent is to accept a plan I_E as input from a planning agent and invoke its services, respecting ordering constraints. The policies for service invocation and error handling are defined by the type of execution agent.

3.4.1 Naïve execution agent

The naïve execution agent receives a plan as input and executes its actions blindly. It ignores errors that occur as a result of communication problems and failed action execution. Table 2 shows the summary of the naïve agent's execution policies.

Table 2 Description of the Naïve execution agent policies using the formalisation described in Section 3.2

Policy	*Description*
Service invocation	Invokes services exactly as described by p_I. The naïve agent requires that $\forall$ actions $a \in p_I$, host(a) $\neq \emptyset$ ^ resources(a) $\neq \{\}$.
Error handling	Ignores execution errors.

3.4.2 Reactive execution agent

The reactive execution agent, like the naïve agent, receives a fully-ground plan as input. The reactive agent, however, can react to action execution failures. Furthermore, if the sensing action detects a fault in plan execution, the reactive agent has the opportunity to isolate and correct the fault.

The method of reaction is based on the type of monitoring. A reactive execution agent working with a synchronous monitor (i.e., analytical) adds its own sensing actions after each action in the original plan. Thus, the reactive agent has some notion of plan execution progress, and if a fault is detected, the agent knows which action(s) caused the fault.

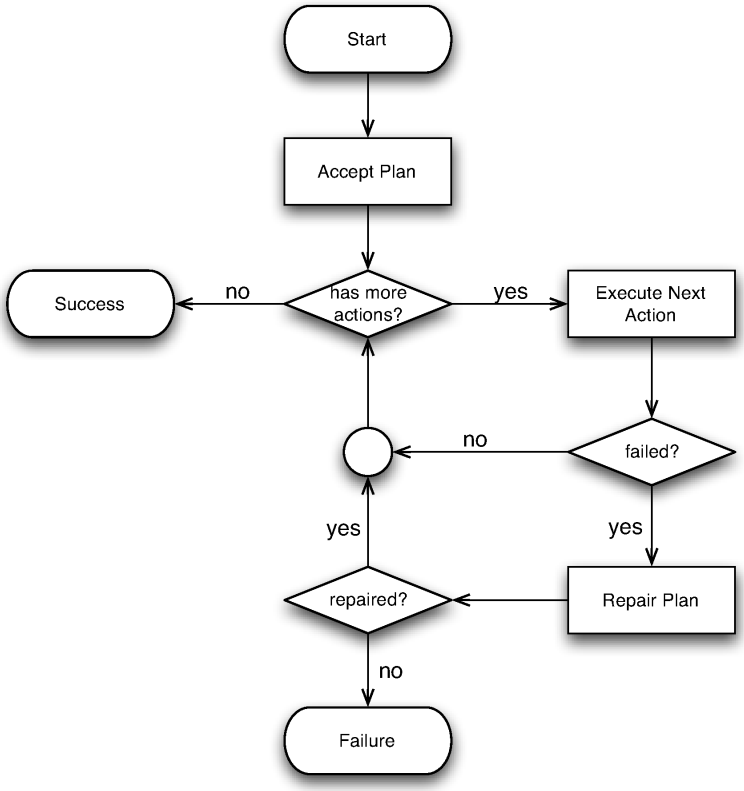
A reactive execution agent working in conjunction with an asynchronous monitor (i.e., data-driven) receives fault information asynchronously. Therefore, the action(s) which caused the fault is uncertain to the execution agent. For this reason, passing fault isolation information to the execution agent is extremely useful for reacting to failures.

Table 3 shows the summary of the reactive agent's execution policies and Figure 6 shows the flow chart of the reactive agent's processing.

Table 3 Description of the reactive execution agent policies using the formalisation described in Section 3.2

Policy	*Description*
Service invocation	Invokes services exactly as described by pI . The reactive agent requires that $\forall$ actions $a \in p_I$, host(a) $\neq \emptyset$ ^ resources(a) $\neq \{\}$.
Error handling	Repairs the failed p_I by replacing failed service call(s) with new ones, creating p_I'.

Figure 6 Flow chart of the reactive execution agent



3.4.3 Proactive execution agent

The proactive execution agent differs from the naïve and reactive agents in that it does not require fully-ground plans as input. Instead, the proactive agent accepts a list of actions and it conducts reasoning to determine how best to allocate resources at execution time. In other words, the proactive execution agent checks for an action's pre-conditions before executing that action, and if those conditions are not met, then it alters the plan (e.g., by modifying or replacing the action).

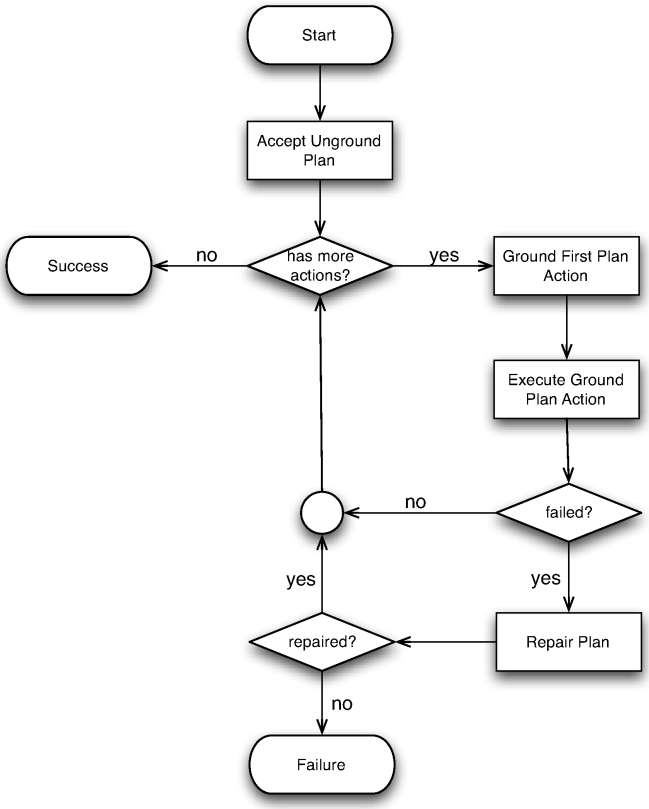
Table 4 Description of the proactive execution agent policies using the formalisation described in Section 3.2

Policy	*Description*
Service invocation	Invokes services using network-aware logic to choose the host and resources at execution time. The proactive execution agent uses only service descriptions from actions $a \in p_I$, meaning $\forall a \in p_I$, host(a) = $\emptyset$ ^ resources(a) = {}
Error handling	Repairs the failed p_I by replacing failed service call(s) with new ones, creating p'_I.

This method requires less-intensive planning, but incurs higher cost at execution time because it conducts resource allocation logic. The proactive agent reacts to failures in plan execution in the same way as the reactive execution agent, although fewer failures are expected to occur due to proactive sensing prior to action execution.

Table 4 shows the summary of the proactive agent's execution policies and Figure 7 shows the flow chart of the proactive agent's processing.

Figure 7 Flow chart of the proactive execution agent



3.5 Network-aware monitoring

Monitoring agents are responsible for monitoring the progress and status of plan execution. The monitoring agent performs FDI, reporting its results to the execution and/or planning agent(s). In this section, three types of network-based monitoring agents are described: the analytical, data-driven, and knowledge-based monitoring agents.

3.5.1 Analytical monitoring agent

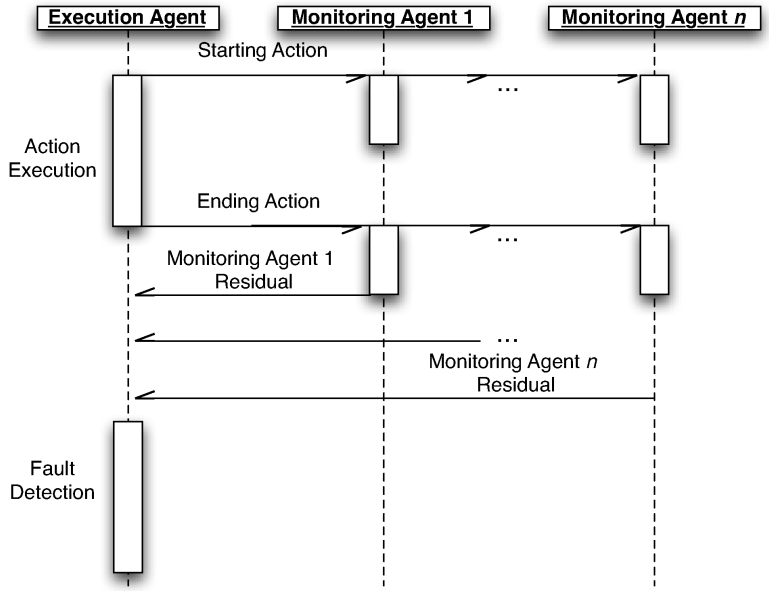
The analytical monitoring agent is a synchronous monitor, based on the model-free observer method. It is synchronous in that the monitor is polled by an external agent. In this method, an action is executed after observing sensor values. When the action

completes, another sensor reading is performed and the observed residual is compared to an estimate.

Analytical monitors are particularly useful to systems where actions have predictable effects on a system. For example, after a crane's grab action, we expect the weight of the crane to be larger than before because it is now holding an item. If the crane is no heavier, than it is arm is empty and we can infer that a fault occurred while executing the *grab*.

In the case of network-centric monitors, we expect actions to have certain effects on the system's network. Network observers monitor the number of packets and bytes transmitted in accordance with each action. A residual is calculated from an estimate based on the action and its resources. Finally, fault-detection logic determines if the residual falls within an acceptable range of the expected value. The sequence diagram for the analytic monitor is shown in Figure 8.

Figure 8 Sequence diagram showing the interaction between the execution agent and analytic monitoring agent(s)



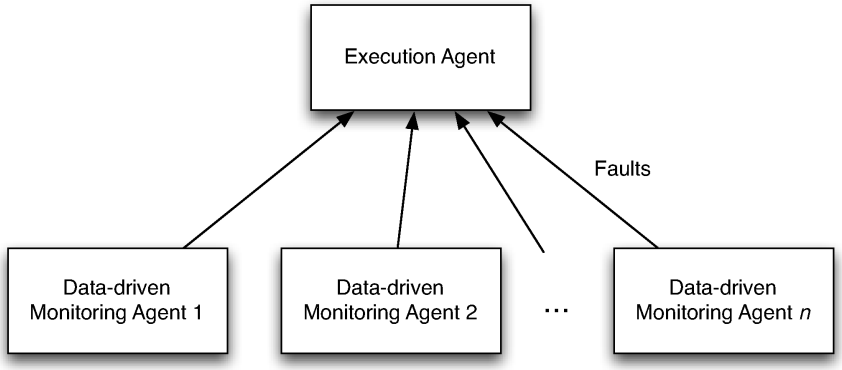
3.5.2 Data-driven monitoring agent

The data-driven monitoring agent is an asynchronous agent that conducts multivariate statistical analysis to ensure that sensor data falls within the limits of a given model. It is asynchronous in that the monitoring agent runs autonomously and informs external agents of faults when they are detected. For this reason, data-driven monitors are useful for systems that operate according to a predictive model.

In our implementation, we monitor network factors that should remain relatively static under normal operation such as packet error rate (PER), the number of retransmitted packets, socket timeouts, connection resets, and failed connections/datagrams.

The sequence diagram for the data-driven monitor is shown in Figure 9.

Figure 9 Flow diagram showing the interaction between the execution agent and data-driven monitoring agent(s)



3.5.3 Knowledge-based monitoring agent

Knowledge-based monitoring agents use advanced reasoning techniques to observe faults in the system. Expert systems and neural networks are two commonly used reasoning techniques in this approach.

In the case of distributed systems, the physical location of the nodes as well as the physical layer and data link layer communication systems have a major impact on the quality of the network. Therefore, geographical location and signal strength can serve as knowledge-based fault indicators.

4 Experiments

This section describes the implementation of the agents and the network emulator in which the experiments were run.

4.1 CORE

All of the experiments in this section were executed in the Naval Research Laboratory's Common Open Research Emulator (CORE) (Ahrenholz et al., 2008). CORE is a framework for emulating networks on a single computer. Using FreeBSD network stack virtualisation, CORE allows for heterogeneous networks to be emulated. Furthermore, the geographical position and mobility of hosts in CORE can be controlled using scenario mobility scripting. Using these features, a large emulation scenario can be deployed and controlled by a single GUI. Figure 10 shows the geographical positions of the hosts and monitoring areas for our IED detection scenario.

All of the core hosts are running simple multicast forwarding (SMF) (Lacharite et al., 2007) for multicast packet forwarding and open shortest path first (OSPF) as a unicast routing protocol. Figure 11 shows a screenshot of the CORE nodes configured to run the IED detection scenario.

Figure 10 Geographical map of the topology of locations, resources, and a network overlay (see online version for colours)

4.2 Network dynamism – mobility models

CORE supports loading *mobility models*, which dictate the geographical movement of hosts while CORE is running. We implemented three mobility models to show how different types of network dynamism affect plan execution and monitoring. All scenarios started with the hosts positioned as shown in Figure 11.

The first mobility model, static, had no host movement. The nodes all remained in the positions shown in Figure 11. The second, dynamic, moved the hosts as dictated by the plan actions that were executed. The last, *partition/merge* or *part-merge*, split the network into two 'islands' (as shown in Figure 12) and merged the network back together. This mobility model caused a complete disconnect between the two islands.

4.3 Experimental process

All planning agents presented were implemented using Tate et al. 's plan-space HTN planner, I-Plan (Currie and Tate, 1991). After the domain and each of the agent types were implemented, each of the planning agents produced plans for five minutes. From these plans, a multi-objective optimisation (MOO) function was used to select a single plan from each of the sets produced.

The plans are passed to execution agents running in CORE. Other hosts in CORE are running the services that correspond to the plan actions. All hosts that house services also run monitoring agents (unless no monitoring agents are specified by the test). The monitoring agents report faults they detect to the execution agent.

Figure 11 Screenshot of the IED detection scenario running in the network emulator, CORE (see online version for colours)

Notes: The blue circles are hosts in CORE and the green lines represent network links. The white rectangles in the background of CORE are used to show the starting locations of the hosts for the IED detection scenario.

4.4 Experimental trials

The variables in the experiments include the following:

- the implementation of the planning agent used
- the implementation of the execution agent used
- the implementation of the monitoring agents used
- the type of network dynamism under which the agents operated.

The types of planning, execution, and monitoring agents we implemented are summarised in Table 5 – those marked with an asterisk are network-aware implementations.

Because CORE uses FreeBSD network stack emulation, different results were experienced on each run – even when all the above variables were constant.

Table 5 A summary of the types of agents and a list of our implementations techniques for those agents

Agent	*Implementation technique*
Planning	Random
	(I-Plan) domain-independent
	Guided*
Execution	Naïve
	Reactive*
	Proactive (sensing)*
Monitoring	Data-driven*
	Analytical*

Notes: Agent implementations marked with an asterisk are network-aware.
* → network-aware.

Figure 12 Screenshot of the partition/merge mobility model running in CORE (see online version for colours)

Note: The blue circles are the CORE hosts, the white rectangles show the starting positions.

Several criteria were used to measure the performance and effectiveness of the combinations of agents and network. For these experiments, we are primarily concerned with the following effectiveness metrics (which are equivalent to our domain-dependent evaluation criteria from Section 3.3.3):

- IED detection accuracy (measured as a percentage of certainty that an IED would be discovered if present)
- Plan execution time (measured as the time required to complete all plan actions).

Also, since we are largely concerned with networking overhead, the scenario's performance was measured using a single metric: the sum of the number of packets sent over each network link. For this metric, a single packet that traverses two network links, (i.e., is forwarded by a network node) counts as two separate packets.

Performance and effectiveness of the trials was judged based on the actions that completed successfully (i.e., the execution agent reported that the plan action was able to be completed, given the available resources and time). For example, manually searching all of the locations yields a 90% IED detection accuracy. However, if a manual search of one area does not complete, the overall IED detection accuracy decreases. Similarly, the execution time depends on the actions executed, the order in which they are executed, and the resources that they utilise. Additionally, during each of the trials, all of the network traffic was logged to determine the network overhead associated with each unique run.

5 Results

5.1 Planning and execution

5.1.1 Effectiveness

When networking is not a factor, as in our control experiment (shown in Figure 13), using the network-aware agents (e.g., the *Guided* planning agent and the *Reactive* execution agents) is of little consequence. All the combinations of planning and execution agents yielded similar execution times.

However, when the network services are distributed, as in Figure 14, network-awareness becomes a critical factor. Note that the network-aware *Guided* planning agent outperforms the *I-Plan* and *Random* planning agents under the *Partition/Merge* network scenario, the most volatile mobility model.

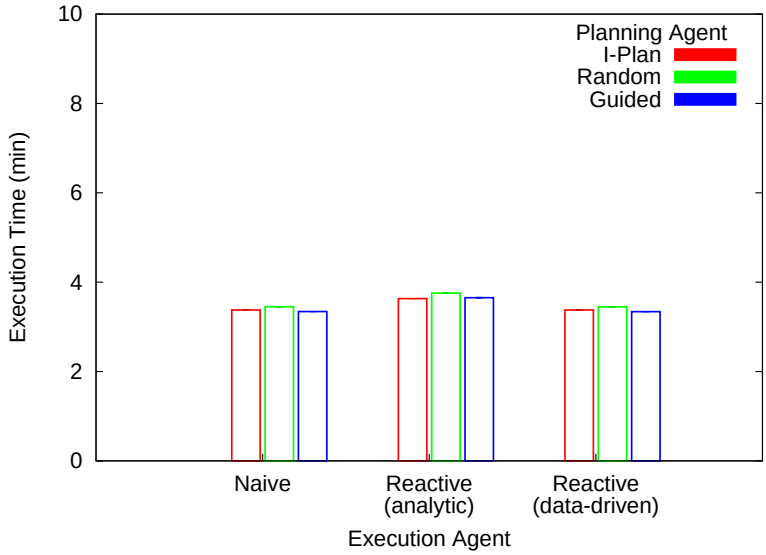
Figure 15 shows the effectiveness of the Naïve, Reactive, and Proactive agents in combination with the I-Plan [Figure 15(a)], random [Figure 15(b)], and guided [Figure 15(c)] planning agents. Effectiveness is measured using two criteria:

1 IED detection accuracy

2 plan execution time.

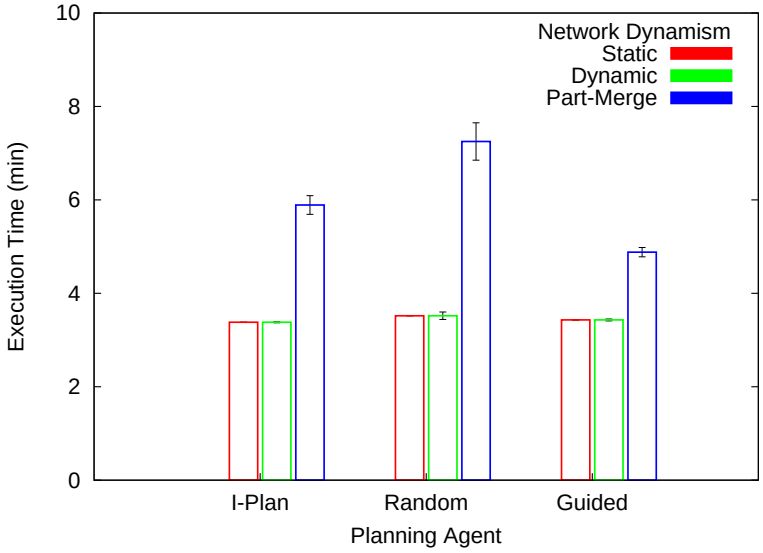
These criteria are plotted along the *x* and *y* axes accordingly – where higher accuracy and lower times are preferred. The results from Figure 15 include equal numbers of experiments using each type of network dynamism (static, dynamic, and partition-merge). We use these results for experimental analysis of the effectiveness of planning and execution agents.

Figure 13 Control experiment: mean plan execution times (in minutes) by plan, execution agent, and monitoring agent types (see online version for colours)



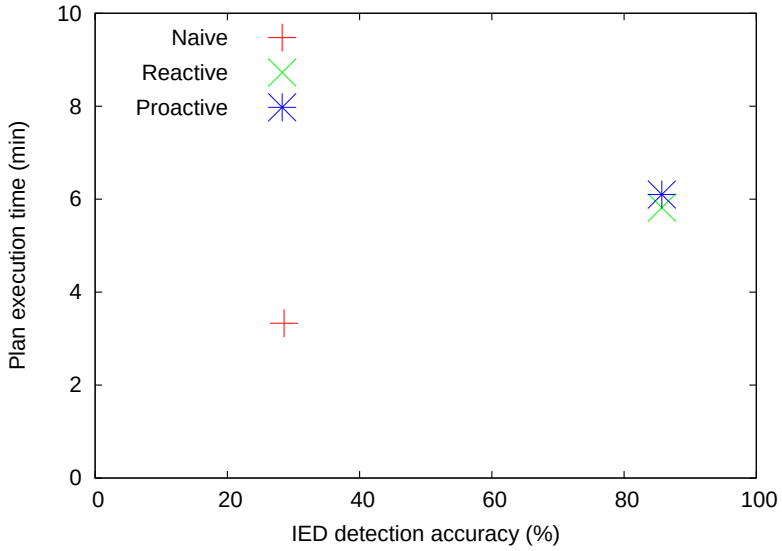
Note: All agents and services are located on a single host so the network is not affecting the results.

Figure 14 Mean plan execution time for plans that executed successfully to completion by planning agent and the network dynamism (as indicated by the mobility scenario) (see online version for colours)

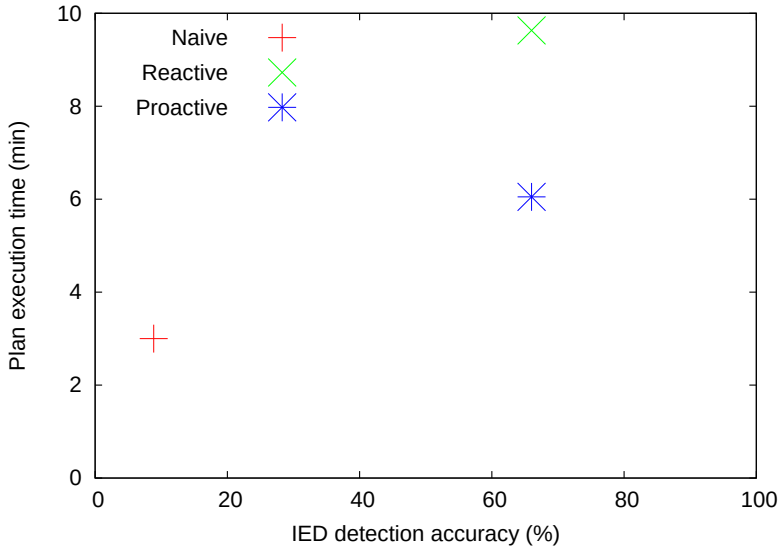


Note: Static mobility exhibits the least network dynamism, dynamic has varying link weights but never completely partitions the network, and partition-merge (part-merge) separates the network into two islands and then merges the islands.

Figure 15 Mean IED detection accuracy versus mean plan execution time of the (a) I-Plan, (b) random, and (c) guided planning agents in combination with each execution agent (see online version for colours)

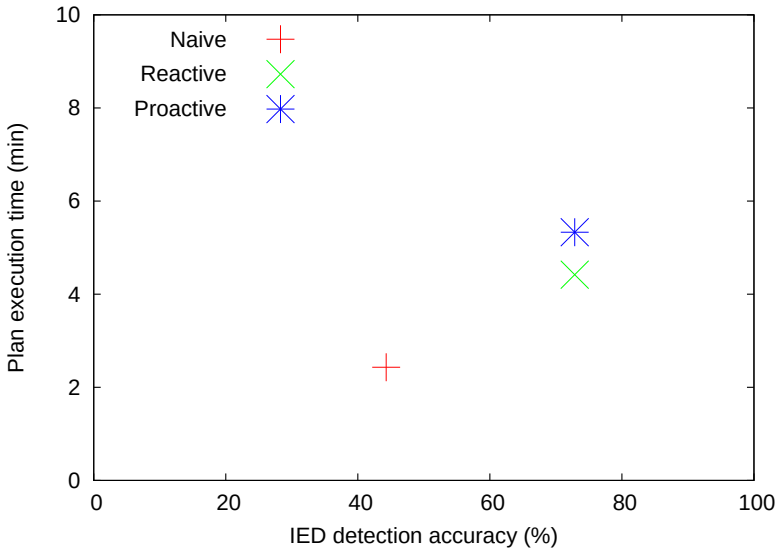


(a)



(b)

Figure 15 Mean IED detection accuracy versus mean plan execution time of the (a) I-Plan, (b) random, and (c) guided planning agents in combination with each execution agent (continued) (see online version for colours)



(c)

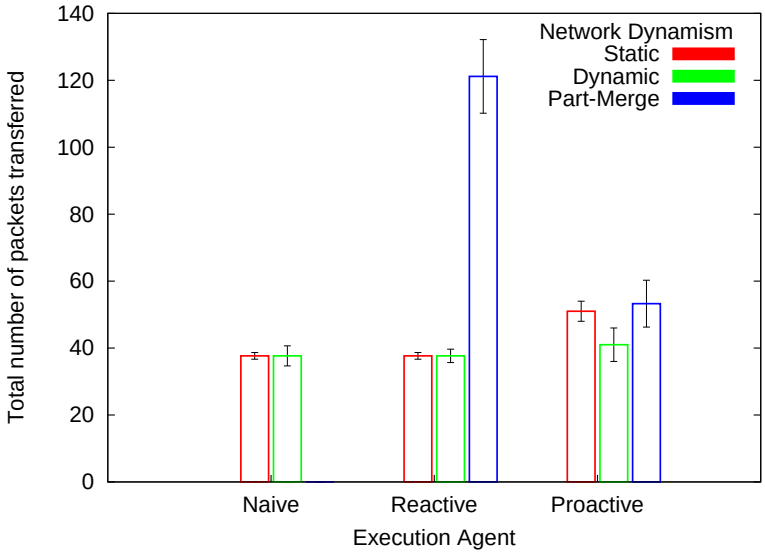
We start with an analysis of the planning agent effectiveness. The I-Plan planning agent [Figure 15(a)] uses domain-independent heuristics (e.g., number of actions in the plan) and IED-domain-dependent heuristics (e.g., IED detection accuracy and plan execution time). Thus, its results tend to be faster and more-accurate than those of the random planning agent. However, the guided planning agent's results further converge on the ideal mix of IED detection accuracy and plan execution time. This is because the guided planning agent also considers the networking conditions during its planning stages. Thus, its plans are usually more-conducive to the network conditions at runtime and perform better than their network-unaware counterparts as a result.

Next, we analyse the results from Figure 15 from the perspective of the execution agent's effectiveness. On average, we find that the naïve execution agent produces faster, but less-accurate, results than the reactive and proactive agents. This is likely because the naïve agent had the greatest percentage of action failures, and contains no logic for recovering from these failures. Another finding is that the reactive execution agent can help improve accuracy over the naïve agent, but can negatively impact the execution time. An example of this negative impact is very clear in the reactive agent's execution time in Figure 15(b). The reason that the execution time can be so large is that the reactive agent simply *reacts* to failures – meaning that it always tries to execute an action, and only after a failure occurs, can it react. The proactive execution agent, on the other hand, anticipates failures so it can improve on the reactive agent's execution time. This improvement comes at the expense of potentially using sub-optimal resources because the resource allocation happens at runtime rather than being planned *a priori*. The system designer must carefully examine the trade-offs between the acceptability of sub-optimal resource utilisation and the large potential gains (around 40% improvement in execution time in our experiments) of proactive network-aware execution.

5.1.2 Performance

Figure 16 shows the average number of packets that were transmitted across all network links over the course of the experiment. The first aspect to note from these results are that the naïve execution agent never successfully completed a plan in the partition/merge network scenario. This is because the naïve agent had no way of dealing with the major network disconnections that occur under that degree of network dynamism.

Figure 16 Average number of packets transmitted across the entire network for various execution agents under different network dynamics (see online version for colours)



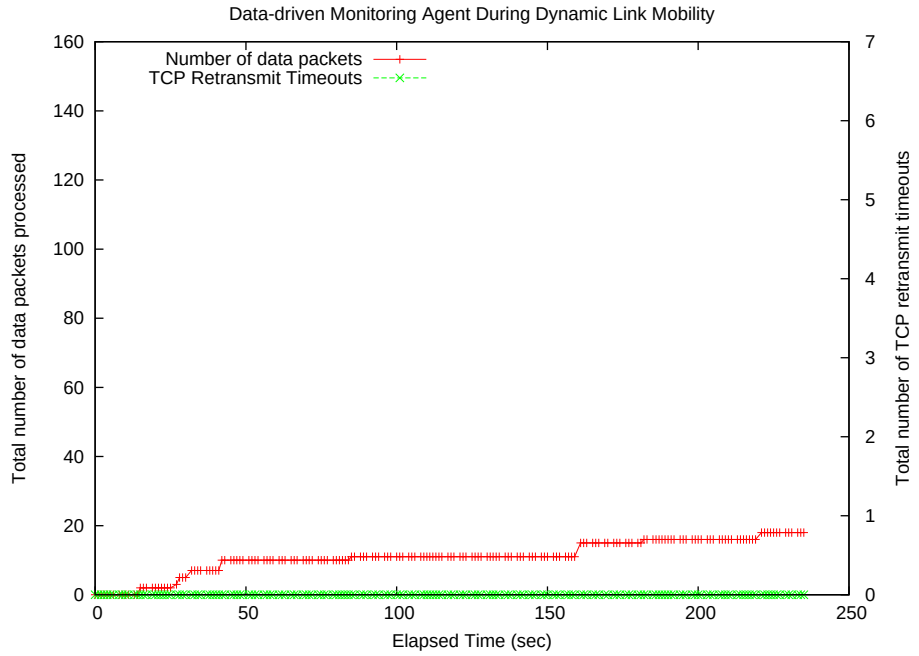
Note: The naïve execution agent never successfully completed executing a plan in the partition/merge mobility scenario, so it is not included in this figure.

The second important aspect of the results in Figure 16 is how the proactive execution agent has a higher constant overhead, but can potentially save on the major network usage that occurs when the reactive agents handle faults. The reason for the proactive agent's overhead is its network probing logic, designed to ensure that the service is available before the agent tries to invoke the service. This small network overhead prevents the major network mis-use that could potentially occur – e.g., the reactive agent in the partition/merge network scenario.

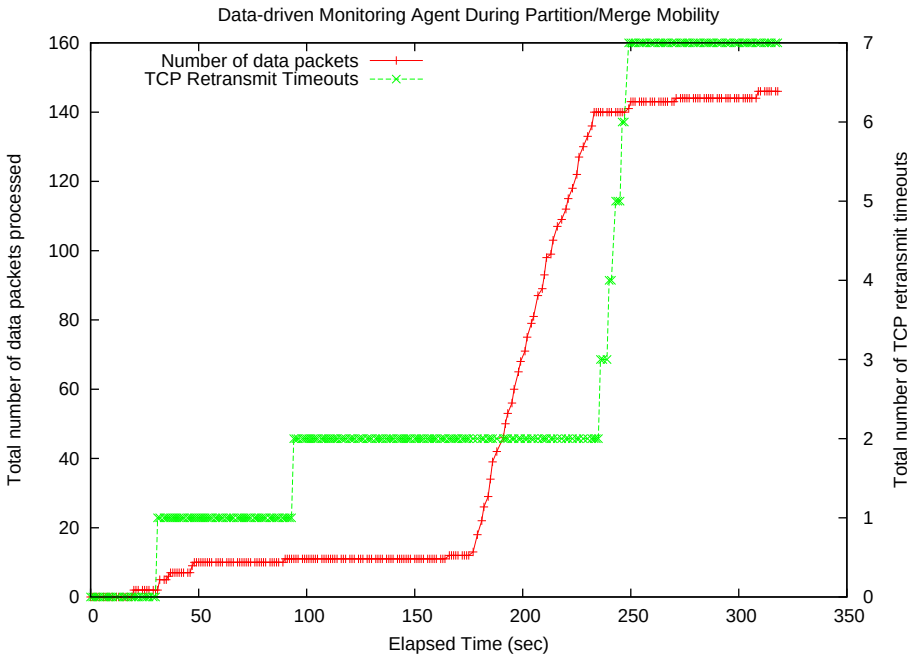
5.2 Monitoring

The analytic monitoring agents triggered a great deal of false positives when there were communication errors (in computing the residual) between the monitoring agents. Also, the analytic monitor was an active monitor, one that has an (possibly detrimental) effect on the system, while the data-driven monitor is passive, having no effect on the system. Furthermore, implementing the distributed residual calculation was more challenging than implementing the data-driven monitor. Thus, we conclude that, in general analytic monitors are less suitable in distributed environments than data-driven monitors.

Figure 17 Network statistics collected by the data-driven monitoring agent during the dynamic link weight mobility scenario, (a) no faults occurred (b) network faults did occur (see online version for colours)



(a)



(b)

The data-driven monitors, furthermore, were very useful as indications of network-related plan execution failures. Figure 17(a) shows the execution of a plan that encountered no failures, whereas Figure 17(b) shows how during a network-failure, the number of retransmitted packets increases while the number of successful outgoing packets remains constant (or near-constant).

Over 54 trials, the data-driven network monitoring agents experienced only a 9.25% false-positive (type I error) rate and a 1.85% false-negative (type II error) rate. False-positives occur more often (five times as frequently) because only one of the monitoring agents has to malfunction for a false-positive to register. For a false-negative to register, a network malfunction must go undetected by all of the monitoring agents.

6 Related work

We first compare and contrast our approach to managing uncertainty in the planning and execution agents. Then, we discuss how our monitoring agents relate to another common approach to monitoring.

6.1 Managing uncertainty

There are several ways to approach the problem of managing uncertainty, e.g., the uncertainty in the IED detection scenario described in Section 3.1. Typical approaches include the following:

- eliminate uncertainty
- reason about uncertainty
- tolerate uncertainty.

Certain types of uncertainty can be eliminated using better hardware or engineering the environment, but this method can increase the cost of planning or lessen applicability to real-world settings. On the other hand, we can reason about uncertainty during planning. This method results in more complex models and therefore higher planning complexity. The last method for decreasing the effects of uncertainty is by tolerating the uncertainty. In this approach, robust plan execution and monitoring help to prepare to handle conditions where plan execution might have otherwise failed. In our approach, we do not attempt to eliminate uncertainty in an effort to maintain practicality in our solution. We do, however, reason about uncertainty and tolerate uncertainty using both our planning and execution agents.

According to Horvitz et al. (1988), "Decision theory is based on the axioms of probability and utility." The purpose of decision theory is to formalise decision making under uncertainty. *Decision-theoretic planning* applies this concept of utilising a probability distribution over the possible outcomes of the actions to prefer plans with a higher *expected* utility. Similarly, *probabilistic planners* use information about the probabilities of the possible uncertain outcomes to construct plans that are likely to succeed. Probabilistic planners, however, typically represent plans as Markov decision processes (MDPs) or use symbolic planning approaches (Kushmerick et al., 1995).

Both decision-theoretic planning and probabilistic planning are done exclusively when a plan is being created. These forms of planning typically add significant complexity to the planning process, making them intractable for execution-time in most cases. These forms of planning typically add significant complexity to the planning process, making them intractable for execution agents. We draw on these decision-theoretic planning in our guided planning agent for plan generation and incorporate less-computationally-expensive forms of uncertainty management in our execution and monitoring agents.

Where decision-theoretic planning reasons about uncertainty at plan-time, *contingency planning* deals with uncertainty by interleaving the planning and execution agents (Wilkins, 1985; Wilkins et al., 1995). Contingency planning most closely relates to our reactive execution agent. The reactive execution agent interleaves planning and execution by conducting on-line plan repairs if an action fails to execute. Contingency planning usually involves tight-coupling between the planning and execution agents, however, our reactive execution agent is loosely-coupled to our planning agents.

In *reactive planning*, no specific sequence of actions is planned in advance. Instead the planner produces a set of condition-action rules, for example, universal plans (Schoppers, 1987) or situated control rules (SCRs) (Drummond and Bresina, 1990). Despite its name, reactive planning most closely relates to our *proactive* execution agent. The proactive agent requires a set of actions, but only grounds the resources of those actions at runtime (rather than plan-time). Thus, reactive planning deals with uncertainty entirely at execution time, as does our proactive execution agent.

6.2 Monitoring

Hart et al. (1990) describes a representation for action progress expectations, which they label *envelopes*. The term comes from performance envelopes in engineering disciplines which describe performance profiles. The point of these envelopes is to avoid wasting time executing costly actions when it is clear that they will fail prior to their completion. Using envelopes allows an agent to:

- modify a failing plan so as to prevent its failure,
- abandon a failing plan,
- retire surplus resources from a succeeding plan,
- improve a plan going unexpectedly well, and/or
- reduce communication between cooperating agents (sharing expectations means they only have to communicate when the expectations are violated).

Envelopes are similar to (and possibly slightly more descriptive than) our monitoring techniques, however we constrain plan modification logic to planning and execution agents. This design decision means that our monitoring agents perform only fault detection, isolation, and identification; and rely on the planning and execution agents to repair the active plan or construct a new plan entirely. We believe this approach better segregates the services provided by each agent.

7 Conclusions

In this paper, we have presented a framework for coordinating distributed IED monitoring and detection in the context of network-centric planning, execution, and monitoring. Furthermore, we have designed and implemented network-aware agents capable of operating in dynamic, heterogeneous networks.

We have demonstrated the importance of network-awareness in a series of empirical analyses based on a realistic IED detection scenario. The agents that exhibit network-awareness outperformed network-naïve agents with respect to not only network-centric criteria (e.g., bandwidth consumed), but also domain-centric criteria (e.g., IED detection accuracy). Further, we find that network-awareness allowed agents to reactively and proactively adapt to highly-dynamic and volatile networks. We believe this network-centric approach is applicable to a broad range of distributed multi-agent systems (MASs).

7.1 Future work

The experiments presented in this work used only two of the three approaches to FDI from Pettersson (2005). Future work will investigate the use of network-aware, domain-specific, *knowledge-based* monitoring agents for further improving the performance and effectiveness of monitoring agents in distributed environments.

Additionally, processing power, like network connectivity, is a limiting factor in the use of distributed services within resource-constrained networks. Future work will focus on incorporating awareness of computational resources into planning and execution agents, thereby enabling our framework to better utilise the processing power of heterogeneous actors within the network.

References

Ahrenholz, J., Danilov, C., Henderson, T.R. and Kim, J.H. (2008) 'Core: a real-time network emulator', *Military Communications Conference*, *MILCOM 2008, IEEE*, pp.1–7, November.

Basili, V.R., Caldiera, G. and Rombach, H.D. (1994) 'The goal question metric approach', *Encyclopedia of Software Engineering*, Vol. 1, pp.528–532.

Chiang, L.H., Russell, E. and Braatz, R.D. (2001) *Fault Detection and Diagnosis in Industrial Systems*, Springer, London, Great Britain.

Currie, K. and Tate, A. (1991) 'O-plan: the open planning architecture', *Artificial Intelligence*, Vol. 52, No. 1, pp.49–86.

Drummond, M. and Bresina, J. (1990) 'Anytime synthetic projection: maximizing the probability of goal satisfaction', in *Proceedings of the Association for the Advancement of Artificial Intelligence*, Boston, MA, pp.138–144.

Hart, D.M., Anderson, S.D. and Cohen, P.R. (1990) 'Envelopes as a vehicle for improving the efficiency of plan execution', in *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, Morgan Kaufmann, pp.71–76.

Horvitz, E.J., Breese, J.S. and Henrion, M. (1988) 'Decision theory in expert systems and artificial intelligence', *International Journal of Approximate Reasoning*, Vol. 2, No. 3, pp.247–302.

Kushmerick, N., Hanks, S. and Weld, D.S. (1995) 'An algorithm for probabilistic planning', *Artificial Intelligence*, Vol. 76, Nos. 1–2, pp.239–286.

Lacharite, Y., Wang, M., Lamont, L. and Landmark, L. (2007) 'A simplified approach to multicast forwarding gateways in MANET', *4th International Symposium on Wireless Communication Systems*, *ISWCS 2007*, *IEEE*, pp.426–430.

Nau, D., Ghallab, M. and Traverso, P. (2004) *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pettersson, O. (2005) 'Execution monitoring in robotics: a survey', *Robotics and Autonomous Systems*, Vol. 53, No. 2, pp.73–88.

Peysakhov, M., Artz, D., Sultanik, E. and Regli, W. (2004) 'Network awareness for mobile agents on ad hoc networks', in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, IEEE Computer Society, Washington, DC, USA, pp.368–376.

Schoppers, M.J. (1987) 'Universal plans for reactive robots in unpredictable environments', in Mcdermott, J. (Ed.): *Proceedings of the International Joint Conference on Artificial Intelligence*, Milan, Italy, Morgan Kaufmann Publishers Inc., San Mateo, CA, USA, pp.1039–1046.

Srivastava, B., Kambhampati, S., Do, M.H. and Nguyen, T. (2006) 'Finding interrelated plans', in *Proceedings of the International Conference on Automated Planning and Scheduling Workshop on Plan Analysis and Management*.

Tate, A. (1993) 'Authority management – coordination between task assignment, planning and execution', in *Proceedings of International Joint Conferences on Artificial Intelligence Workshop on Knowledge-based Production Planning, Scheduling and Control*.

Wilkins, D.E. (1985) 'Recovering from execution errors in SIPE', AI Center Technical Note 346, SRI International, January.

Wilkins, D.E., Myers, K.L. and Lowrance, J.D. (1995) 'Planning and reacting in uncertain and dynamic environments', *Journal of Experimental and Theoretical Artificial Intelligence*, No. 7.1, pp.121–152.