Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

A CASE STUDY OF THE DEVELOPMENT AND USE OF A MANA-BASED FEDERATION FOR STUDYING U.S. BORDER OPERATIONS

Emmet R. Beeker III
Ernest H. Page

The MITRE Corporation
7515 Colshire Drive
McLean, VA 22102, U.S.A

ABSTRACT

A federation approach is used to expand the geographic extent of MANA (Map Aware Non-uniform Automata), a cellular-automaton based agent simulation, in order to support a study of investment strategies for border protection along a portion of the southern U.S. border. The federation is implemented using the Department of Defense (DoD) High Level Architecture (HLA). Federation performance is optimized using HLA Data Distribution Management (DDM) services and through a bypass of the normal HLA mechanisms for ownership transfer. Analysis of the running federation indicates that overhead due to federation processing is minimal – less than 6% of the total federation runtime (94% of the runtime is due to processing in the MANA simulations) .

1 INTRODUCTION

Agent-based simulation emerged from the Artificial Intelligence community as a class of discrete event simulation. (Some might argue that agent-based simulation is a rediscovery and repackaging of discrete event simulation, but that's an article for another time). Official definitions of the concept vary throughout the literature, but the primary factor that causes a discrete event simulation to be an agent-based simulation seems to be the notion that the entities in the simulation model represent real-world entities that possess facilities for cognition and learning (Edmonds and Möhring 2005).

Agent-based simulation has been used in a variety of contexts: modeling individual and group behaviors in financial markets, road networks, military combat, epidemiological systems, and so forth. A good overview of the use of agent-based simulations appears in (Sanchez and Lucas 2002).

In 2005, U.S. Office of the Border Patrol (OBP) initiated work on a set modeling capabilities to facilitate asset and portfolio management within the Douglas Station of the United States. A prototype capability based on the MANA (Map Aware Non-Uniform Automata) simulation was developed by the MITRE Corporation that illustrated how tradeoffs between investments in technology (e.g., sensors), tactical infrastructure (e.g., fences), and personnel could be studied and evaluated. The success of that prototype led to a request from OBP to extend the model to support analysis across a larger expanse of terrain, the Tucson Sector.

In this article we describe the development of the Tucson Sector model. Section 2 provides a brief overview of MANA. The implementation of terrain in MANA precluded a direct extension of MANA to encompass a region the size the Tucson Sector. An approach to federate a collection of MANAs using the Department of Defense (DoD) High Level Architecture (HLA) was selected to accomplish this geographic scaling. The basic federation design is described in Section 3. Some runtime optimizations based on HLA Data Distribution Management, and modifications to the HLA ownership transfer scheme are described in Sections 4 and 5, respectively. Conclusions and areas of future study appear in Section 6.

2 MANA

MANA is a cellular automaton based model of military conflict developed by the New Zealand Defence Technology Agency, and is one of the suite of models supporting data farming for the U.S. Marine Corps Project Albert (USMC Project Albert 2006). MANA was designed for use as a scenario-exploration model primarily to facilitate study in areas where traditional "physics-based" models fare poorly, e.g., representing and reasoning about the effects of enhanced situational awareness and command and control.

MANA explores the interaction of autonomous entities (agents). Each agent has "personality traits" that drive it toward or away from other agents on the battlefield, and these agents may change their personality due to events in

the model. Agent behaviors are governed by four types of parameters (Galligan, Anderson, and Lauren 2004):

- *Personality weightings* determine an agent's propensity to move towards friendly or enemy units, towards a waypoint, towards easy terrain, and towards a final goal point.
- *Movement constraints* modify personality weightings. The *cluster* parameter turns off an agent's propensity to move towards friends above some maximum cluster size. The *advance* parameter prevents an agent from moving towards an objective without a minimum number of friendly units accompanying it. The *combat* parameter determines the minimum local numerical advantage a group of agents require before approaching the enemy.
- A third set of parameters describes the basic capabilities of the agents, such as weapons and sensors, movement speeds and interactions.
- A final set of parameters provides options on the movement characteristics of the agents, including things like whether terrain affects speed, the degree of randomness when moving and if obstacles should be avoided.

MANA behaviors are specified for both individual agents, and groups of agents known as *squads*. A squad is a group of agents of any size. These agents share the same properties, and can switch between states either individually or as a group. A *state* is a set of parameter values that determine the agent's current behavior. Apart from behavioral and capability parameters, entities in the same squad also share both Situational Awareness (SA) of enemy contacts, and waypoints that may be used to guide the agents around the battlefield.

The default battlefield for MANA is a 200x200 grid of cells, each of which can be occupied by a single live entity (except where multiple agent occupation of cells is explicitly allowed). Cell types include: (1) billiard table, (2) easy going; (3) wall; (4) light bush/dense bush; and (5) hilltop. Agents are repelled by the playbox boundaries; Agents cannot wander off the battlefield.

The application of MANA to support a wide variety of military analyses is described in numerous articles and technical reports, including the annual *Maneuver Warfare Science* compendium, available from the Project Albert web site (USMC Project Albert 2006).

3 MANA FEDERATION DESIGN AND IMPLEMENTATION

To support studies involving the Douglas Station, the MITRE Border Patrol simulation project developed a MANA simulation configured with a playbox of 1000 x 1000 cells (pixels). Each cell represents an area of 2500 square meters, so that the entire playbox covers an area of 50 kilometers by 50 kilometers. This 50 km resolution was sufficient for initial studies involving the Douglas Station, however, the Tucson sector covers approximately 420 kilometers along the U.S.-Mexico border.

Options for increasing the size of a single instance of MANA from 50 km to 420 km include: (1) increasing the number of cells in the playbox, while leaving the cell resolution fixed at 50 meters; or (2) increasing the size of the cells, while leaving total the number of cells fixed. Increasing the number of cells beyond 60,000x60,000 is infeasible in MANA, because the application depends upon the indices being 16 bit numbers. Increasing the number to 2000x2000 increases the memory usage for the playbox by a factor of four, and increases the processing time similarly. Using a single 1000x1000 cell MANA instance to represent 420 kilometers would require that each cell represent a space of 420x420 meters. Since a great deal of "interesting activity" could happen within a 420 meter space, using a single MANA instance to represent the entire Sector is undesirable.

A third alternative to achieve geographic scaling is to appeal to distributed simulation – partitioning the terrain across fourteen instances of 30 km MANA (Figure 1). Processing for the additional cells would be in parallel, minimizing impact on total running time.

Figure 1: Geographic Overlay for MANA Federation.

The DoD High Level Architecture (HLA) provides a standard for constructing distributed simulations, which are typically called *federations* in HLA parlance. Successful applications of the HLA in the DoD context abound. One of the primary technical challenges faced in the development of a MANA-based federation is efficiently overcoming language incompatibilities. Existing commercial and

government HLA Runtime Infrastructures (RTIs) provide C++ and Java interfaces. MANA is written in the Borland Delphi variant of Object Pascal. Pascal's object structure, data structure, and variable types do not match corresponding C++ Structures, so a C++ RTI library cannot be directly linked and utilized by Object Pascal code. Similar incompatibilities are present between Object Pascal and Java.

For this effort, we adopted the Virtual Technology Corporation's RTI NG-Pro which provides a C++ interface. To overcome this barrier, our approach implements a dynamic link library (DLL) which acts as a translator from Pascal to C++.

3.1 MANA to RTI Adapter

HLA provides an RTI Ambassador object with methods that an application calls to send information to the distributed simulation. HLA expects the application to provide a Federate Ambassador object, implementing the methods that are called by the distributed simulation middleware. MANA is implemented in Borland's Delphi Programming Language (DPL) as a Windows application. DPL is an evolution of Borland Object Pascal. It maintains many of the Pascal conventions so it is not directly compatible with C++. While it might be possible to determine the name-mangling necessary for MANA to call services in the RTI library, it would be extremely difficult to implement the Federation Ambassador callbacks to MANA. Since DPL allows specification of different calling styles, we construct a DLL that presents a 'C' interface to MANA and implement the Federation Ambassador and RTI Ambassador interfaces needed by the RTI DLL. Our DLL, referred to as the *MANA Adapter*, is implemented with Microsoft Visual Studio.

When a DLL is loaded by an application, its initialization code is executed. During initialization, Microsoft prevents any other DLL from loading. This means that the MANA Adapter DLL cannot reference objects that are initialized in the RTI DLL. In particular, the MANA Adapter DLL cannot instantiate a Federation Ambassador nor RTI Ambassador object during DLL initialization. Instead, the MANA Adapter must use reference pointers and create the ambassadors during a call after initialization.

MANA declares and implements five functions for the MANA Adapter to call:

1. mEntityUpdate() – receives agent status from MANA Adapter;
2. mEntityCreate() – creates a new agent in MANA when an agent moves into the play-area;
3. mEntityDelete() – deletes an agent that moves out of the play-area;
4. mSquadActivate() – activates a squad in MANA when the first agent in the squad moves into the play-area;
5. mSquadDeactivate() – deactivates a squad when the last agent in a squad leaves the play-area.

The MANA Adapter has methods to register the MANA callbacks, based on the number and type of parameters. For example, mEntityDelete(), mSquadActivate(), and mSquadDeactivate all have a single parameter of integer type, so they are registered in the same method of the MANA Adapter:

```
void (__stdcall *mEntityDelete) (long int
pEntityID);
void (__stdcall *mSquadActivate) (long int
pSquadNumber);
void (__stdcall *mSquadDeactivate) (long int
pSquadNumber);

void rRegisterSingle( int procNumber, void
(__stdcall *p) (long int li) )
{
    if (procNumber == 1)
        mSquadActivate = p;
    else if (procNumber == 2)
        mSquadDeactivate = p;
    else if (procNumber == 3)
        mEntityDelete = p;
}
```

The function pointer is passed in as a parameter. It was found that the integer parameters to callbacks must be declared as *long int* in the C++ DLL and as *Integer* in MANA in order for the call stack to be properly recognized.

MANA calls six functions in the MANA adapter:

1. rInit() – creates the Fed Ambassador and RTI Ambassador objects;
2. rJoinFederationExecution() – tells the MANA Adapter which federation to join and which federate this is;
3. rRegisterObjectInstance() – causes the MANA Adapter to register an entity object with the RTI and receives an object handle;
4. rEntityUpdate() – sends updated agent status to the MANA Adapter;
5. rNextEventRequest() – tells the MANA Adapter that MANA has finished this time step and is ready to process the next time step;
6. rResignFederationExecution() – tells the MANA Adapter to resign from the federation and clean up allocated memory.

During simulation initialization, MANA registers each agent with the MANA Adapter. MANA refers to each agent by its index number in an agent array. The MANA Adapter registers the agent with the RTI and receives a

global handle for the agent. The MANA Adapter keeps a table of the MANA index and the RTI handle for each agent.

3.2 MANA Federation Protocols

The Tucson Sector covers approximately 420 kilometers along the U.S. – Mexico border. This requires a geographic partitioning across a minimum of fourteen 30 km MANA instances (see Figure 1). Recall that within standalone MANA, agents are unable to travel beyond the perimeter of the MANA playbox. But, agent migration across MANA instances is essential to the representation of reasonable behaviors in the MANA federation. Modifying the way MANA handle perimeters, though, requires significant changes to MANA. One of the goals of the MANA federation effort is to minimize changes to MANA, and provide the majority of the functionality necessary for federation operations within the MANA Adapter.

To ameliorate edge conditions, while minimizing code changes to MANA, another thirteen MANA instances are added which overlap the fourteen primary MANA instances. This results in a federation of 27 MANA instances (Figure 2). Solid purple squares denote primary MANAs. Dashed green squares denote overlapped MANAs that support agent migration.

Figure 2: Geographic Overlay for MANA Federation

Within a given time step, MANA examines each agent, one at a time in random order, to determine the agent's action. Using a set of rules and preferences and based upon its situational awareness, each agent will move or shoot (remember, MANA was designed to be a combat model; for use in a Border Patrol context the weapons ranges are small and the act of "shooting" represents alien apprehension). The action of each agent is determined by the conditions within the playbox: nothing external to the playbox affects the agent. Generally, agents in the MANA border simulation are influenced by the state of cells at a distance between 25 and 100 cells from the agent. An agent near a playbox boundary cannot know about anything outside the playbox – in the adjacent MANA instance. In order to allow an agent to have situational awareness across MANA instances, we overlap MANA instances by 50%, as shown in Figure 2.

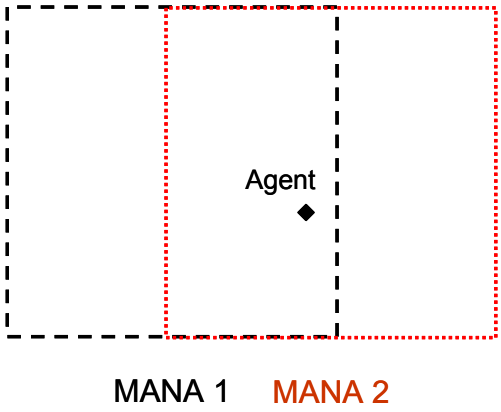

Figure 3: Overlapped MANA Play Areas

Each agent in the overlapped region exists in two MANA instances simultaneously. During its local processing, each MANA instance determines the agent's action in its playbox and reports the action to its local MANA Adapter. But an agent will always be closer to an edge in one MANA playbox and closer to the middle of the playbox in the other MANA instance. In Figure 3, the agent is closer to the edge in MANA 1, and closer to the middle in MANA 2. The MANA with the agent closer to the middle is considered the controlling MANA and its action is deemed to be the "correct" action. In this way, an agent in the controlling MANA has at least 250 cells between it and the perimeter. In Figure 3, MANA 2 is the controlling MANA.

As each MANA instance determines the action of an agent, it updates the agent status and it reports the new status to its MANA Adapter using the rEntityUpdate() procedure. If the MANA is the controlling MANA, then the MANA Adapter will report the agent status to the RTI. Note that the MANA instance does not know it is the controlling MANA. It reports the status of all agents to the MANA Adapter. The MANA Adapter is responsible for determining control.

The RTI sends the agent status to any subscribing federate. In the case shown in Figure 3, the MANA Adapter for MANA 2 sends the agent status to the RTI and the RTI sends the agent status to the MANA Adapter for MANA 1.

After a MANA has updated all of its agents, it tells the MANA Adapter that the time step is finished by calling the rNextEventReuest() procedure and waiting for the return. As part of the processing for rNextEventRequest(), the MANA Adapter calls the RTI nextEventRequest(). This allows the RTI to send to the MANA Adapter all of the agent status messages received from other MANA Adapters. Status updates for agents in the playbox are sent to MANA through the mEntityUpdate() procedure which overwrites the agent status determined by the MANA instance. If the agent is not yet known to the MANA Adapter, thus not known in the MANA instance, mEntityCreate() is called to create the agent in the MANA instance. If necessary, mSquadActivate() is called first to make sure there is squad data for the entity. If an agent has moved out of the play area, the mEntityDelete() procedure in MANA is called, and the mSquad Deactivate() procedure will be called if there are no more agents from the squad in the play area.

If the position of the agent is in the central area of the MANA instance (horizontal cell 251–750), then the MANA Adapter acquires ownership of the attributes of the agent as described in Section 5.

4 OPTIMIZING DATA DISTRIBUTION

While production runs used 27 MANA federates running for 7,200-10,800 time steps, we used fewer federates and fewer time steps during development. Some production runs took nearly four hours, which was too long for our development cycle. Therefore, we executed the federation for 1,000 time steps for federations of 5, 7, 11, and 19 federates. The federation executed on dual dual-core opteron computers. The five MANA federation executed with three MANAs on one computer and two MANAs on another computer. The HLA RTI Executive and Federation Executive ran on its own computer. For the other federations, the first three MANAs executed on one computer, and the other computers each executed four MANAs. Figures 4 and 5 show the average execution time per time step for the first 1,000 time steps. Because the execution time per time step increases in MANA, this number cannot be used to calculate run time for more or less time steps.

Execution time for the MANA federation can be divided into three parts: (1) MANA processing time, (2) MANA Adapter and RTI processing time, and (3) message transit time (network latency).

MANAs execute in parallel, but they process each time step synchronously. Since the agents may be distributed non-uniformly across the terrain, MANA processing times are uneven. MANAs that finish processing for a given time step must wait for the MANAs that are still processing. The last MANA to finish executing a time step determines the total processing time. The HLA prevents other MANAs from processing status update messages until the last MANA finishes its own internal processing.

While network latency is not a factor for the MANA federation, filtering and processing messages is. There are 5500 agents in a typical MANA federation execution. If the distribution of agents is more-or-less uniform, then each MANA sends approximately 200 status update messages per time step. With no optimization, each of the messages is sent to all of the other federates via an exploder mechanism in the RTI. This results in each MANA processing approximately 5300 messages per time step. But each MANA is only interested in the messages from the MANA that overlaps it on the left and on the right; representing only about 400 of the 5300 status messages it receives.

The HLA provides for data distribution management (DDM). Using DDM, MANA Adapters can subscribe to updates for a region of the state space and they can declare which region an agent is in. By creating point regions, the RTI establishes channels between the communicating MANA Adapters, and only sends messages to the appropriate MANAs. Thus each MANA Adapter only receives approximately 400 messages from its overlapping neighbors at the end of each time step.

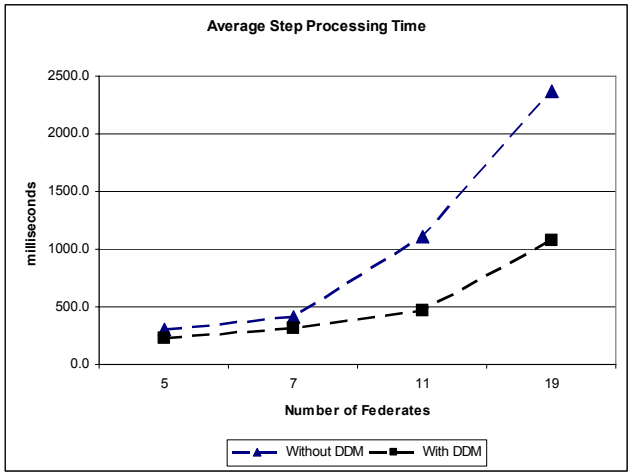

Figure 4: MANA Step Time Using DDM

Figure 4 shows the improvement using DDM. Without DDM, every MANA must send agent data for every agent to every other MANA. The total number of messages increases as the square of the number of federates. Each MANA must process approximately 200 messages per distributed federate. With DDM, messages are only sent to the overlapping MANA. Each MANA receives about 200 messages from the left overlapping MANA and 200 messages from the right overlapping MANA. As the number of federates increases, the benefit for using DDM

increases, so that at nineteen federates, the improvement is slightly better than fifty percent.

5 OPTIMIZING OWNERSHIP TRANSFER

In addition to the state update messages sent between MANA instances each time step – which are minimized through the use of DDM – ownership transfer negotiations occur as agents cross MANA instance boundaries. Ownership transfer requires several steps in the RTI: the requesting MANA Adapter calls the RTI which, in turn, calls the MANA Adapter owning the agent's attributes. The owning MANA Adapter must release ownership, call back to the RTI, and the RTI will call back to the requesting MANA Adapter with notification that it now has ownership. Ownership transfer happens asynchronously, unlike attribute value updates. In order to synchronize ownership transfer, an *acquireOwnership* interaction is used in addition to the HLA-provided ownership request. MANA receives the interaction before a time step only when all of the other MANAs have finished processing the preceding time step. Thus it synchronizes ownership transfer.

Ownership transfer is required because the RTI enforces ownership rules for attribute value updates. Only the federate that owns an entity is allowed to send messages with values for the entity's attributes – called *reflect* in HLA. The spirit of HLA is that the owning federate reflects attribute values and that if another federate wants to change an attribute value, then that federate must acquire ownership. However, HLA provides another method of sending state information, called interactions. Interactions are intended for one-time occurrences. Because interactions are intended for *events other than attribute value changes* there are no ownership rules for interactions. By creating an *updateAgentStatus* interaction, we are able to substitute interaction parameters for attribute values. This arguably violates the intent of HLA, but allows a significant performance improvement.

This change allows the federation to omit all of the HLA ownership procedure calls. MANA Adapters still send the ownership interaction. However, they can assume they have ownership as soon as they send the interaction. Because the RTI uses reliable delivery and the interaction is time-stamped, it is guaranteed that the owning MANA receives the interaction before the next time step. When a MANA Adapter receives an ownership interaction, it knows that the sending MANA "owns" the agent and will be sending the status update for the next time step.

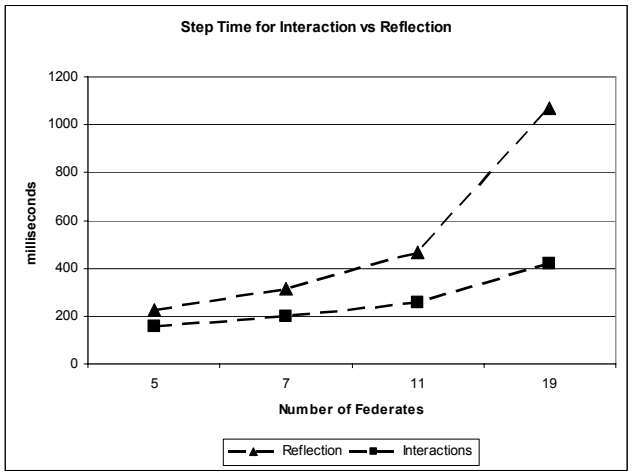

Figure 5: MANA Step Time Using HLA Interactions

Figure 5 shows the improvement due to using interactions. With nineteen federates, execution time is less than half when using interactions, compared to reflects. Execution time is less than twenty percent of the original execution time.

Through the combination of DDM and a removal of explicit ownership transfer calls, processing time in the twenty-seven MANA federation is reduced from 90 minutes to 60 minutes for a 1200 time step scenario, and the total time spent in the MANA Adapter and RTI is reduced from nearly 30 minutes to less than three minutes.

6 CONCLUSIONS

Through a relaxation (or reinterpretation) of its intrinsic concepts (e.g., shoot = capture) an agent-based simulation developed to study military combat may be successfully used to study issues relevant to the U.S. Border Patrol. Interoperability technology such as the High Level Architecture (also originating in a military context) provides a convenient mechanism to expand the geographic extent of a cellular-automata model such as MANA without significantly modifying the model itself. We were able to take a 50km x 50km Station model and extend it to a 420km x 30km Sector model in just a few man-months of effort. The extensibility of this approach to a national-level model, however, is unclear. For such a geographic extent, the number of federates would likely become unwieldy – a significant amount of infrastructure was developed to support launching, running and monitoring scenarios. For a national-level model, we are investigating changes to the underlying MANA terrain implementation, or an implementation in an alternative agent framework.

ACKNOWLEDGMENTS

The work described here was conducted as part of The MITRE Corporation support to the U.S. Border Patrol. The opinions expressed are those of the authors and do not reflect official positions of The MITRE Corporation or the U.S. Border Patrol.

The MANA federation benefited greatly from the efforts of Brian Pridemore and Woan Sun Chang. We are also grateful for the project leadership of David Brooks and Paul Wehner. A very special thanks to Annette Wilson, of Virtual Technology Corporation. Her assistance—especially for navigating RID settings, DDM, and other RTI minutiae—was invaluable.

REFERENCES

Edmonds, B. and M. Möhring. 2005. Agent-based simulation modeling in social and organizational domains. *Simulation*, **81**(3):173-174, March.

Sanchez, S.M. and T.W. Lucas. 2002. Exploring the world of agent-based simulations: simple models, complex analyses, In: *Proceedings of the 2002 Winter Simulation Conference*, 116-126.

Galligan, D.P., M.A. Anderson, and M.K. Lauren. 2004. Map Aware Non-uniform Automata, Users Manual, Ver. 3.0, July.

USMC Project Albert. <www.projectalbert.org> [accessed February 11, 2006].

AUTHOR BIOGRAPHIES

EMMET R. BEEKER III is the Senior Technical Advisor for the Modeling and Simulation Group, Center for Acquisition and System's Analysis at The MITRE Corporation. He holds a B.A. in Mathematics and M.Sc. in Computer Science from Indiana University. His research interests include simulation-based analysis and parallel discrete-event simulation. He is a member of the IEEE Computer Society and of the ACM. His e-mail address is <ebeeker@mitre.org>.

ERNEST H. PAGE is a member of the technical staff for The MITRE Corporation. He received the Ph.D. in Computer Science from Virginia Tech in 1994. He serves on the editorial boards of ACM *Transactions on Modeling and Computer Simulation*, SCS *Simulation*, SCS *Journal of Defense Modeling and Simulation*, and the *Journal of Simulation*, and is the ACM SIGSIM representative to the WSC Board of Directors. His e-mail address is <epage@mitre.org>.