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.

SECURITY CHECKPOINT OPTIMIZER (SCO): AN APPLICATION FOR SIMULATING THE OPERATIONS OF AIRPORT SECURITY CHECKPOINTS

Diane Wilson

Transportation Security Administration
WJHTC, Bldg. 315
Atlantic City International Airport
Atlantic City, NJ 08405 USA

Eric K. Roe
S. Annie So

Northrop Grumman Corporation
12011 Sunset Hills Road
Reston, VA 20190 USA

ABSTRACT

For most security planners, a key challenge is to continuously evaluate how changes or additions to their facilities or procedures impact security effectiveness, operational costs, and passenger throughput. Each change must be analyzed to ensure negative effects do not outweigh the benefits. This paper presents Security Checkpoint Optimizer (SCO), a 2-D spatially aware discrete event simulation tool developed by Northrop Grumman for the Transportation Security Administration (TSA), a part of the U.S. Department of Homeland Security. SCO is designed to allow security analysts to graphically build a simulation model and layout a series of screening activities to take place. Once the model is defined, SCO simulates passenger movement using both path-based and pathless movement algorithms to mimic a semi-autonomous passenger traversal of a 2-D space. The software is designed to allow analysts to perform multiple "what-if" analyses to balance benefits and tradeoffs.

1 BACKGROUND

In the post-9/11 era, security has been in the spotlight at all of the nation's airports. There exists the ongoing need to improve security posture to meet government mandates and ensure safe and uninterrupted operations while simultaneously ensuring rapid and efficient passenger flow.

Security planners must continuously consider how procedural and technological changes or additions affect their operational costs, the satisfaction of the traveling public, and security. In anticipation of rapid changes and improvements in transportation security technology, the TSA's Transportation Security Laboratory (TSL) Modeling and Simulation Program and the National Safe Skies Alliance (NSSA) initiated the research and development of an easy to use, spatially aware, discrete event simulation tool that would support analyzing performance impacts of new passenger screening equipment and protocols for the nation's transportation security checkpoints in a timely and efficient manner. The result was the *Security Checkpoint Optimizer* (SCO) application.

2 OVERVIEW OF THE SCO SOFTWARE

The Security Checkpoint Optimizer is a Java-based application capable of helping security planners better understand their current and proposed operations. SCO allows the analyst to graphically build a simulation model that can be used to compute results for

- Passenger and bag throughput
- Security effectiveness (Probability of detection)
- Resource utilization
- Operational cost

The primary driver for the above computations is a custom-built, spatially aware, discrete event simulation engine. This engine is embedded in a rich-client interface, which allows the analyst to layout the scene using a combination of drag-and-drop and simple point-and-click drawing tools.

The following sub-sections describe some of the key application concepts the analyst must understand to successfully build a model using the SCO software.

2.1 Entities

Entities are model elements that are created dynamically during a simulation run. They typically represent the actors or participants in the simulation model (e.g., passengers). Entities are not added to the model by the user, but rather by a special schedule element called a *generator*. Once created by the generator, entities are placed into the scene. The entities traverse the scene by moving between different areas via traversable model elements such as *paths* and *regions*.

Traversal of the model by entities is one of the most fundamental and important aspects of using the SCO to model real-world, spatially-oriented processes.

2.2 Traversable Elements

A *traversable element* is a model element that supports traversal by an entity. In the context of the SCO, "traversal" refers to the process by which entities are moved through the scene by way of predetermined or dynamically generated routes. *Paths*, *regions*, and *action points* are types of traversable elements.

To support entity traversal, traversable elements must be able to generate or support the generation of routing instructions for the entity to follow. Paths have very simple routing calculations to perform — the entity is instructed to follow the path exactly. Regions must perform more complex calculations because each entity may have many valid routes through the region. In addition, the region must ensure that entities maintain spacing stand-off constraints and do not collide. Under normal circumstances, the SCO user does not have to concern themselves with the route generation components of the system.

2.2.1 Paths

A *path* is a strictly defined route between two points in the scene. Paths are unidirectional; there is one point of entry to the path and one point of exit. The point of entry is the *start* and the point of exit is the *end*.

Each path is represented visually as a multi-segmented line with endpoint icons. The endpoint icons denote the path start (a triangle) and end (a circle).

One of the most important capabilities of a path is its ability to enforce a queuing regime in a part of the model.

2.2.2 Regions

A *region* represents an open area without any prescribed route through the area. Each region is represented visually as a multi-segmented polygon. Regions are used to model areas such as lobbies, atriums, wide halls, or other areas where entities are able to move in multiple directions.

Regions provide the SCO software with one of its most important features — the ability for the **software** to determine the routes entities take rather than forcing the modeler to prescribe all possible routes through the system. This capability has several benefits: it can significantly reduce the time to create and/or modify a complex model; and it can aid in understanding the effects of congestion on the movement of entities. Lack of such a "pathless-movement" capability in reviewed commercial-off-the-shelf simulation packages (Kukulich et al. 2000) was one of the primary drivers in TSA's decision to build SCO.

2.2.3 Action Points

An *action point* is a type of traversable element which has no height or width — as the name implies, it consists only of a single point. Because action points have no area, the system allows only a single entity to occupy the action point at any given time. Action points are often used to model activities taking place at well-defined locations by a single entity. They are used in lieu of small regions because they are somewhat faster to place in the model.

2.3 Modifiers

A *modifier* is a type of model element applied (attached) to another model element that changes that element's default behavior. The model element to which the modifier is being applied is referred to as the modifier's *target* or, alternatively, the *target element*. Not all model elements will accept a modifier.

A modifier itself has no direct visual representation in the scene, but application of a modifier to a particular target may cause the target to alter the way it displays itself. For example, a region may change its color if a specific type of modifier is applied.

Activities, filters, traits, and tokens are types of modifiers.

2.3.1 Activities

An *activity* is a type of modifier whose purpose is to simulate a real-world process at some location in the scene. Activities take place in a user defined sequence upon and entities transition into a traversable element such as a path or a region. The activities that take place on a path or a region are collectively referred to as a *behavior*. In some cases, a behavior is implemented by a single activity; in other cases, a behavior may be implemented by several coordinating activities. Each activity also has an associated (optional) *conditional expression* that can be used to decide if an activity should be executed or skipped.

SCO includes a core set of activities that are very general-purpose (Move, Dispose, Acquire) and several checkpoint-specific activities such as Drop Bags and Scan.

2.3.2 Filters

A *filter* is a type of modifier whose purpose is to restrict the flow of entities into a traversable element. Filters answer the basic question, "is it valid for the entity to enter?" and are the primary means for the modeler to specify and control the routes entities take to traverse the scene. For example, a filter might be used to restrict entry of passengers onto a path based on attributes such as the passenger's selectee status or class (first or coach). Multiple filters

may be applied to a traversable element and all must evaluate to true before an entity is allowed entry.

2.3.3 Tokens

A *token* is a modifier that can be used to collect statistical and other run-time data from entities as they make their way through the modeled scene. Unlike the other modifier types, tokens are not placed on target elements at design time, but rather at run-time by an activity. SCO supports the token types listed in Table 1.

Table 1: Token Types

Type	Description
Count	A token used to count the total number of times issued. Each time a count token having a given name is issued, the count is incremented by one (1). A count token would be used to count the number of entities passing through a specific part of the model.
Event	A token that records when an action of note occurs. An event token can be used to generate demand statistics.
Duration	A token that records both a begin and end time. A duration token can be used to compute timing statistics.

In addition to their usefulness in collecting data, tokens can also participate in routing decisions via filtering based on whether or not an entity possesses a particular token.

2.3.4 Traits

A *trait* is a modifier that can be used to add new properties to a visual model element at design-time. The primary reason to use a trait is to give a visual model element one or more properties that other elements in the model may use. The key point here is that when a trait is applied to a particular visual model element, that element will not necessarily "know" what data the new property is intended to convey.

A good example of a trait is the *Entry Point* trait. The Entry Point trait is applied to a traversable element to inform a generator that the traversable element can serve as a point of entry into the modeled scene. The traversable element which is being modified by the trait knows nothing of the trait's purpose — it's the generator that knows how to interpret the trait.

2.4 Other Model Elements

The SCO software also includes several other types of model elements that are commonly found in discrete-event simulation packages. These elements include

- Resources,
- Conditions,
- Generators,
- User-defined variables, and
- Uniform and non-uniform random number generators.

3 EXAMPLE TWO-LANE SECURITY CHECKPOINT MODEL

This section provides an introductory overview of many SCO modeling concepts. It is intended to quickly give the reader some familiarity with the SCO, but does not contain sufficient details to be considered a complete introduction to the software.

Figure 1 shows a notional two-lane security checkpoint model. It consists of two x-rays, two walk-through metal detectors (WTMD), two hand-search stations, an explosive trace detection (ETD) station, a number of demarcated areas (the colorful polygons), and paths (lines that start with a little triangle and end with a circle).

When constructing a model, it is common to import some form of background which provides spatial context; this is often a CAD drawing in .DXF (Drawing Exchange Format) format. SCO supports using DXF files as well as JPEG and GIF files as a background image. For the purposes of this discussion, the background consists of the metal detectors, x-rays, hand search tables, and ETD. While visible in the scene, these objects do not affect the way the simulation executes — that's the job of paths, regions, and other model elements.

The spatially important logical elements in Figure 1 are the traversable elements, which include 12 regions and 6 paths. Probably the easiest way to understand the model is to describe its runtime behavior.

When the user chooses to run the model, it is the job of a *generator* to create and initialize the entities that will participate in the model run. In the case of a typical checkpoint model, there are two types of entities: passengers and baggage. The following paragraphs describe the life-cycle of the passenger entity with only passing references made to carry-on baggage.

At ❶ the generator has created a new passenger entity and placed it into the scene by means of a *transition* into a region. The generator knows that the leftmost region is a valid entry point by virtue of a *trait* applied to the region.

Once placed into the scene, the passenger entity begins performing its *activities*. Each traversable element can supply one or more activities for entities to perform. These activities represent the behavior of the passenger, and it is the job of the modeler to decide what behavior must occur at each location; however, to speed the model building process, traversable elements provide a set of default activities. Regions and paths include a Move, Transition, and

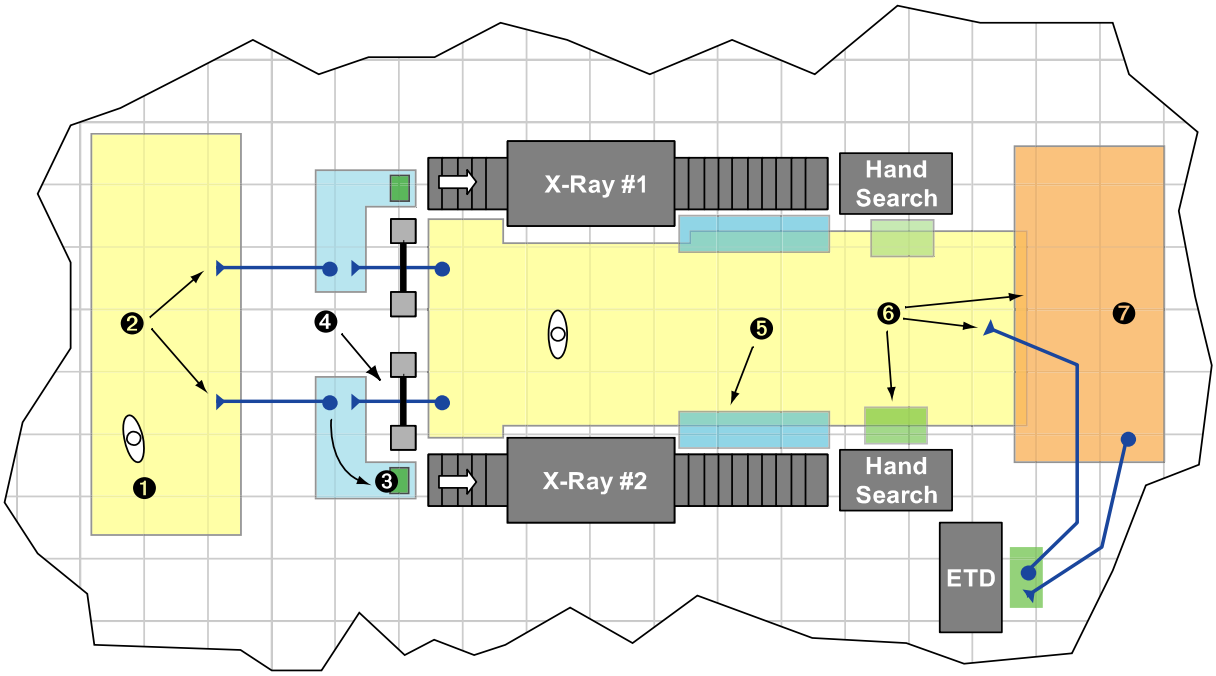

Figure 1: Two-Lane Security Checkpoint (Notional)

Abandon activity. Action points include a Transition and Abandon activity. Of interest to this example are the Move and Transition activities.

The Move activity does two things: (1) it attempts to locate another traversable element where the passenger entity may transition into; and (2) it starts the passenger movement toward the selected traversable. In the case of Figure 1, the passenger entity at ❶ has two choices to leave the entry region: they may choose to move on to one of two paths ❷ which represent the queues into the checkpoint lanes. Choice of which path to take may depend on such things as queue length or the passenger's distance from the entry to the path; but for this example, it is assumed neither path receives any preferential treatment. With no preference, the simulation runtime will randomly select between the two paths.

Once the passenger entity arrives at the start of their selected path, the Move activity terminates and the Transition activity executes. The Transition activity moves the entity logically from the region to the path. This is a key distinction between Move and Transition: Move moves the position of an entity spatially, while Transition moves the entity logically from one traversable element to another. After transitioning, a new set of activities commence — a process that continues until the entity is removed from the scene via a *Dispose* activity.

Continuing the example, suppose the bottom path was selected. After moving along the path, the passenger entity eventually arrives in another region (the L-shaped region just in front of the WTMD and X-Ray #2). There are actually **two** ways out of this region: (1) take the path through the WTMD; or (2) move into the small region located just in front of the x-ray in-feed conveyor belt. To prescribe a particular choice, the modeler makes use of two filters. In this case, the desired behavior is to allow only those entities with carry-on bags to proceed to the small bag-drop region, and only those without carry-on bags to proceed along the path through the WTMD. To accomplish this, a filter is placed on the bag-drop region ❸. This filter uses the *conditional expression*

```
hasCarryonBags > 0
```

to indicate only a passenger whose `hasCarryonBags` property is greater than zero, may use the bag-drop region. Another filter is placed on the path through the WTMD ❹. This filter uses the conditional expression

```
hasCarryonBags = 0
```

to indicate only a passenger whose `hasCarryonBags` property is set to zero, may use the path through the WTMD.

At runtime, if a passenger entity has any carry-on bags, the passenger will first proceed to the bag-drop region. Then a *Drop Bags* activity will ensure any carry-on bags are taken from the passenger entity and deposited at the appropriate x-ray in-feed. After divesting their bags, the passenger can use the path leading through the WTMD.

Once the passenger has proceeded through the WTMD, they enter another large region. At this point

there are several exit possibilities. If the passenger has dropped bags, they will proceed to collect their bags at ❺. The passenger will use the correct bag-pickup area based on a property given to them when they dropped their bags at the x-ray and a filter that examines the property. This combination of filter and property essentially says, "if you used x-ray #1 when you dropped your bags, pick them up at x-ray #1's pickup area; otherwise use x-ray #2."

At ❻ the passenger will determine whether they can move into the exit area or must proceed to hand search or the ETD station. Again, this selection is based on filters and properties. For example, a passenger with the property `hasLaptop` might be required to proceed to the ETD.

Finally, once the passenger makes it into the exit region ❼, a Dispose activity will remove the passenger from the simulation, collecting any statistics from the passenger entity before it is deleted.

4 ANALYSIS

SCO analysts can now create virtual checkpoint and perform "what-if" sensitivity analyses to balance cost, level of service and security. Some of the typical analyses performed include designing checkpoint equipment configurations within limited physical space, and evaluating impacts of new policies and procedures.

4.1 Physical Space and Checkpoint Configuration

Physical space can have significant impact on screening equipment configuration and subsequently passenger flow. SCO's spatially aware features are designed to address the physical space concern which is commonly lacking in most discrete event simulation tools.

SCO allows the analyst to move equipment and expand functions easily. All the checkpoint processes and procedural components are embedded in the traversable elements. Analysts can easily change the DXF background, then drag-and-drop or adjust the size of traversable elements to accommodate the new layout without changing the underlying logic in the model. In most cases, analysts can also duplicate functionality or equipment by copying and pasting the traversable elements in the scene. Using similar concepts, analysts can design multiple security checkpoints and layout check-in counters, kiosks, and lobby checked bag screening functions to study the entire facility operation. Once the model is redefined, SCO will handle the movement and collision avoidance for passengers as well as baggage. Users can observe the 2-D animation to identify potential bottlenecks or trouble areas for further refinement. Figure 2 shows a notional airport lobby environment including the passenger check-in activities, lobby checked bag screening and security checkpoint operations.

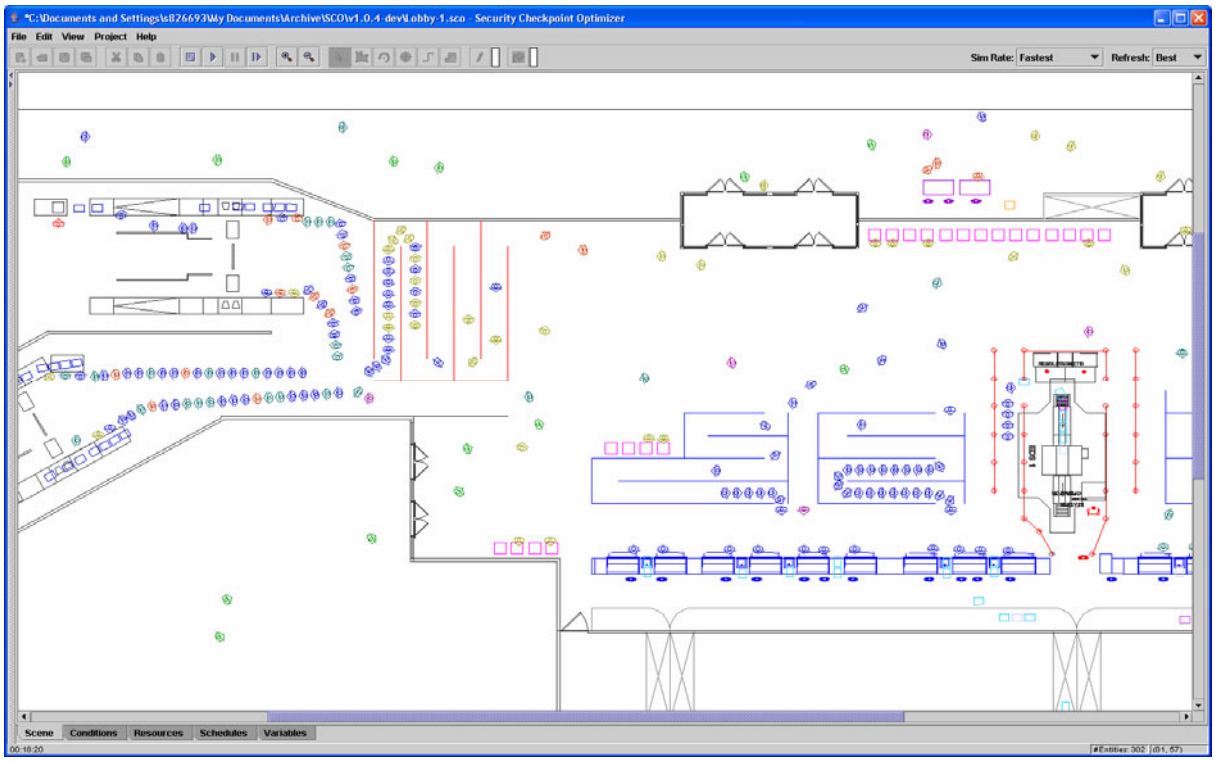

Figure 2: SCO Running a Notional Airport Lobby Scene Including Check-in Activities, Lobby Checked Bag Screening and Security Checkpoint Operations

4.2 Checkpoint Procedural and Policy Change

Frequently, security analysts use SCO to evaluate security and passenger flow impacts of new procedural and policy changes. Some of the recent TSA policy changes such as increasing random carry-on item searches or passenger pat-down searches, and the change in the prohibited item list are simple adjustments to the variables in the model. Using reporting tokens and resources, analysts can produce quantitative results and recommendations to support policy makers in their decision making process. Other experiments that SCO is designed to answer include staffing levels and requirements, impact of future checkpoint technologies, concept of operations, security effectiveness, and whole facility security design and evaluation.

4.2.1 Checkpoint Performance Forecast

Analysts can load passenger arrival data file(s) to the SCO passenger generator and estimate the performance of the checkpoint as well as the rest of the operations throughout the course of the study duration. This is particularly helpful in studying the peak performance at each operation, evaluate the level of service (i.e., wait time, queue length) and optimize the staffing requirements among all the facility security operations.

4.3 Security Effectiveness

As entities flow through the modeled scene, SCO allows the analyst to evaluate the security effectiveness of the model. This is accomplished by aggregating probabilities of detection for some item of interest ($p_{d(i)}$) at key points during the model traversal. Each $p_{d(i)}$ may be based upon lab or field testing, input from subject matter experts, or gathered through analysis of data accumulated over time.

$$P_d = 1 - \prod_{i=1}^{n} \left(1 - p_{d(i)}\right)$$

The final probability of detection (P_d) can then be collected from each entity as it exits the model. Different probabilities can be computed for each item of interest (e.g., guns, knives, explosives, etc.).

Using the security effectiveness data, analysts can identify the weakest routes through the system and respond by changing screening equipment and/or procedures. They can also examine how such changes intended to improve security affect system throughput.

4.4 Multi-level Transportation Security Applications

TSA uses a set of Standard Operating Procedures (SOP) to establish uniform processes and standards for providing security screening services. Parallel to the SOP, security analysts can create and define a set of standard SCO concept of operations (conops) models to support internal continuous improvement, and estimate staffing requirements, as well as budget requirements. The same standard SCO models could be made available by TSA to security planners and officers to support layout of their checkpoint and develop virtual operations for holiday level of service projections, staffing estimates, as well as emergency planning and preparedness.

At the current time, SCO software developers are implementing additional features such as real time data store, Design of Experiment run manager, scripting language, and passenger screening specific activities to enhance the tools and better support ongoing efforts within TSA.

5 LEVERAGING THE INTERNET

In August 2005 development began on a new user-interface for SCO simulations. The concept was to provide an application that could use the SCO discrete-event engine while presenting a simplified interface that was accessible via the Internet. Since SCO is written entirely in the Java programming language, Java WebStart was selected to serve as the launch protocol for this new application, called the Network-Enabled Security Checkpoint Optimizer or *NetSCO*. Figure 3 shows the output page from notional NetSCO Checkpoint model — a model very similar to the one discussed in this paper.

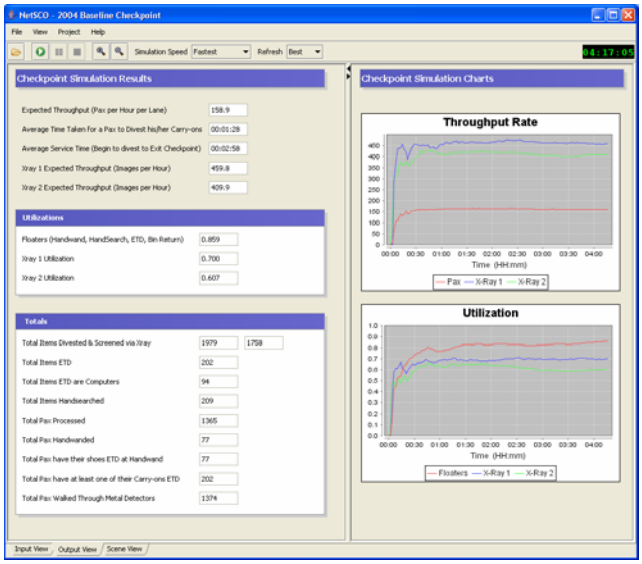

Figure 3: Notional NetSCO Checkpoint Model

Several open-source packages were utilized to simplify implementing visually appealing forms-style pages and charting. These packages include JGoodies (<https://jgoodies.dev.java.net>), Abeille

(<https://abeille.dev.java.net>) a forms designer for JGoodies, and the JFreeChart charting library (<http://www.jfree.org>).

NetSCO is easily started via a secure web browser. Its central data store helps to better manage configuration control of available models as well as facilitating updates to the data and supporting libraries.

At the current time, NetSCO is available to select TSA personnel.

6 FUTURE DIRECTIONS

Development of SCO and NetSCO is ongoing. Several additions and enhancements to SCO are under investigation including:

- Converting the user-interface to the Eclipse Rich Client Platform.
- Developing a more powerful built-in reporting capability on top of the Eclipse Business Intelligence Reporting Tools (BIRT).
- Enhancing and optimizing the performance of the open-area movement algorithms.
- Incorporating the Mozilla Rhino JavaScript scripting engine to allow the analyst to script complex behaviors.

7 SUMMARY

SCO is a spatially aware discrete event simulation tool designed to address physical space concerns, passenger behavior, and passenger movement incorporated with the traditional queuing model methodology to solve today's transportation security challenges. It's graphical, easy-to-use interface allows security planners to quickly assemble and analyze a series of security screening operations. By leveraging NetSCO, analysts can build high fidelity models and customize the user interface for decision makers to evaluate potential policy impacts. Research of SCO was initiated by TSA's Transportation Security Laboratory Modeling and Simulation program and the National Safe Skies Alliance. SCO and NetSCO are being used to support a number of TSA planning and analysis activities, and are undergoing continuous enhancement and improvement as part of several ongoing TSA initiatives.

REFERENCES

Kukulich, M., C. Schuhmacher, E. Roe, and M. Fertal. 2000. *Evaluation of Commercially-Available Software for Airport Security Modeling*. U.S. Department of Transportation, Federal Aviation Administration. Unpublished.

AUTHOR BIOGRAPHIES

DIANE WILSON is a computer scientist with the Department of Homeland Security, Transportation Security Administration, located at the William J. Hughes Technical Center (WJHTC), Atlantic City, NJ. Ms. Wilson is part of the Systems Engineering Branch with responsibilities that include Modeling and Simulation (M & S) Projects Lead and requirements development. Prior to being appointed the M & S Projects Lead, Ms. Wilson worked as the Infrastructure Protection Product Area Lead. Before joining the Aviation Security Laboratory (now the Transportation Security Lab) in February 2000, Ms. Wilson worked for 2 years as a test engineer and 6 years in operational support as a systems engineer/software engineer/computer specialist and training coordinator, providing testing and software/operational field support for the Voice Switching and Control System (VSCS), an Air Traffic Control (ATC) communications system. Ms. Wilson earned a certification from the Project Management Institute (PMI) as a Project Management Professional (PMP) and has earned a Master's Degree in Aeronautical Science from Embry-Riddle Aeronautical University, a Master's Certificate in Project Management from George Washington University (ESI), a Bachelor of Science Degree in Computer Science from the Richard Stockton College of New Jersey, and an Associate Degree in Applied Science in Business Data Processing from Cumberland County College. Ms. Wilson is a member of Phi Theta Kappa and the International Test and Evaluation Association. Her e-mail address is <diane.wilson@dhs.gov>.

ERIC K. ROE is a software engineer with Northrop Grumman Corporation and has been working with and developing simulation applications for twelve years. He has worked on behalf of customers including the U.S. Air Force, Federal Aviation Administration, Transportation Security Administration, and Food & Drug Administration. Mr. Roe has developed simulations in C++, Java, Python, and C#. He holds a BS in Computer Science from the State University of New York at Binghamton. His e-mail address is <eric.roe@ngc.com>.

S. ANNIE SO is an operations research analyst with Northrop Grumman Corporation and provides analytical and simulation modeling support to customers including Department of Homeland Security, Transportation Security Administration and Transportation Security Laboratory. She holds a BS and MS in Industrial and Systems Engineering from Virginia Polytechnic Institute and State University, and has been working in industrial engineering and simulation for twelve years. She is a certified project management professional. Her e-mail address is <annie.so@ngc.com>.