Process Query Systems for Surveillance and Awareness

Vincent Berk Wayne Chung Valentino Crespi
George Cybenko Robert Gray Diego Hernando
Guofei Jiang Han Li Yong Sheng
Thayer School of Engineering
Dartmouth College, Hanover NH 03755

ABSTRACT

Many surveillance and sensing applications involve the detection of dynamic processes. Examples include battlefield situation awareness (where the processes are vehicles and troop movements), computer and network security (where the processes are worms and other types of attacks), and homeland security (where processes are terrorist financing, planning, recruiting and attack execution activities). A Process Query System is a novel and powerful software front-end to a database or real-time sensing infrastructure that allows users to define processes at a high level of abstraction and submit process definitions as queries. We describe a current working implementation that has been used for vehicle tracking using an acoustic sensor network and for computer worm detection.

Keywords: Tracking, Surveillance, Process Query Systems.

1. INTRODUCTION

Process Query Systems (PQS) are software systems that allow users to interact with multiple data sources, such as traditional databases (DBS) and real-time sensor feeds, in new and powerful ways. In traditional DBS's, users specify queries expressed as constraints on the field values of records stored in a database or data recorded by sensors, as allowed by SQL and its variants for streaming data. By contrast, PQS's allow users to define processes and to make queries against databases and real-time sensor data feeds by submitting those process definitions. A PQS parses the process description and performs sequences of queries against available data resources, searching for evidence that instances of the specified process or processes exist.

A major innovation of the PQS concept is the virtual machine that it presents to the applications programmer. Our experiences with several application areas over the past few years suggest that PQS can improve the effectiveness of data-intensive applications programming by abstracting away and standardizing much of the programming logic and detail that is presently often painstakingly recreated in low-level implementations. Moreover, the discipline of process oriented thinking that PQS's impose on applications developers will help develop more powerful and robust data-intensive applications more quickly.

Our efforts in designing and implementing a PQS are at the exploratory stage. While an alpha version has been implemented and tested with encouraging results, significantly more basic research needs to be conducted to develop a robust, general-purpose PQS. We believe this technology can have a significant impact on several homeland security challenge areas such as: distributed infrastructure monitoring and management, plume detection by an array of sensors, vehicle tracking, worm detection and frequent itemset discovery in counter-terrorism data mining.

In Section 2 we provide a description of the general architecture of our PQS. Then, in Sections 3 and 4, we present two different examples, the former based on the classical problem of tracking vehicles in an Euclidean space, and the latter based on the problem of tracking active worms in the Internet. An overview of work in progress concludes the paper.

2. GENERAL ARCHITECTURE

The process query system consists of three major components: User Interface, TRAFEN tracking and fusion engine, and Message-Oriented Middleware. Distributed sensor networks sense and acquire data on the battlefield or within a computer network. A set of fuselets apply signal processing and other logic to the raw sensor data to form observation messages or events, e.g., e.g. tank observed at $72.01/-43.02$ (longitude/latitude). These messages are published into the message-oriented middleware under specific topic headings, e.g., ground vehicle. On the other side, the front-end interface allows users to define process models with high-level abstractions such as Hidden Markov Models (HMM) [1], The process model, as well as a set of message topics, are submitted to the back-end TRAFEN tracking and fusion engine. TRAFEN parses the process model and subscribes to those messages, available from the middleware, that match the user-specified topics. These messages could originate from sensors in the battlefield, from intelligent databases, from human observers, or from many other sources. Multiple Hypothesis Tracking (MHT) algorithms are invoked with the user-defined process model to determine the hidden relationships among the observa-

tions and to build hypotheses representing the true state of the observed system. Hypothesis results and predictions can be used to support decision-making processes. Moreover, they can be published back to the middleware as message sources for other fusion engines. Sensor networks can also subscribe to these hypotheses and predictions as feedback to optimize their sensing tasks.

User Interface: Users select message subscription topics and define process models via the user interface. They can browse the topics published on the MOM and subscribe to the specific topics of interest, and define process models using various high-level abstractions. A process model describes the state transition of a tracked target, which evolves with time according to specific known laws such as the kinematic constraints on the target. State-transition models and their encompassing process definitions can be described at a high-level of abstraction with state equations, Hidden Markov Models (HMM), rule-based models, and so on. Process definitions can be described with high-level of abstractions with these models. For example, with a graphical interface, users can use standard templates to define the states and state transitions of an HMM process. The process model, its parameters, and the event subscription topics are formulated into a process query that is submitted to the back-end TRAFEN engine.

Message-Oriented Middleware (MOM): Messaging is a method of communication between software components or applications. MOM is a middleware communication mechanism that allows different, loosely coupled applications to communicate with each other in an asynchronous, connectionless way. The sender and the receiver do not have to be available at the same time to communicate via MOM, nor do they need to know anything about each other except the message format. There are two general types of messaging: message queuing and publish-subscribe. Message queuing is a point-to-point communication model, while publish-subscribe provides delivery to more than one receiver at a time. Publish-subscribe is particularly useful when many receivers need the same data, when high performance is required, or when the data has to be delivered in real-time. Conceptually, messages are clustered in the MOM into specific topic groups or queues. In our framework, we are building a semantic MOM based on Sun's Java Messaging Service. Both topics and message contents are marked up with DAML [2] so that they can have embedded semantics. Sensor fuselets push their observation messages to the MOM under specific topic headings. On the other side, the TRAFEN engine receives all messages within the same topic from the MOM and starts the multiple hypothesis tracking process. By clustering ad-hoc messages according to their topic headings, the entire set of targets and measurements are divided into independent groups. Instead of solving one large tracking problem, a number of smaller problems are solved in parallel. Therefore, we can manage the size of the hypothesis space and the computing complexity of each MHT algorithm.

TRAFEN: TRAFEN is designed as a generic fusion engine to implement multiple hypothesis tracking/fusion processes. After TRAFEN receives vehicle observations from the MOM, for example, it uses Reid's multiple hypothesis tracking algorithm [3] and Bayesian formulations to determine the probabilities of alternative associations of observations to tracked targets, each possible association forming a unique hypothesis (or sequence of related events). Different data-association algorithms are used to cluster events in different problem domains. Multiple target tracking algorithms, such as Reid's algorithm, recursively calculate the likelihood that a new observation is associated with each existing hypothesis. Then the new observation is added into one or more hypotheses according to the ranked likelihoods, and the set of hypotheses is updated. The process model specified in the query is used to compute the conditional probability that an observation is associated with the existing hypotheses. For vehicle observations, a Kalman filter [4] (or other least-squares methods) may be used to predict the vehicle state at a future time.

The TRAFEN engine consists of several main components: MHT algorithms, prediction models, hypothesis management and storage, and event subscription/publication. Various MHT algorithms are implemented as modules in our engine architecture. Every engine also includes a pool of standard prediction-model implementations. As we mentioned above, the process query includes parameters that are the inputs to the related prediction models. The MHT algorithms will be invoked with the instantiated model and the message inputs from the MOM. Hypothesis management involves several maintenance operations. Observations not assigned to existing tracks initiate new tentative tracks; a tentative track becomes a confirmed track upon satisfying some quality tests; and low-quality tracks, as determined by the update history, are deleted.

3. GROUND VEHICLE TRACKING

In this section, we present an original Java implementation of Reid's classical Multi-Hypothesis Tracking algorithm [3]. The original MHT algorithm was designed to handle incrementally a ranked set of hypotheses, each hypothesis consisting of a set of consistent tracks of observations. Reid distinguished two sensor groups that he called Type I and Type II respectively. Type I sensors, such as radars, are able to provide information on the number of targets in the area being scanned, while type II sensors, such as acoustic sensors, do not have this capability. This difference affects the way observations are processed. Type I sensors return groups of observations that are supposedly taken at the same time (e.g., a radar scan), while Type II sensors return single observations.

Our current implementation of the MHT algorithm assumes that data are produced by fuselets based on Type II sensors. Thus, single observations are processed one at a time and added to existing tracks, or are used to define new tracks or are treated as noise or clutter. We are upgrading our software, however, to cope with Type I sensors as well.

One of the typical problems in the definition and mainte-

nance of multiple hypotheses of consistent tracks is that, in the absence of information about the kinematics of the moving targets, the set of likely hypotheses grows exponentially. This is due to the fact that each new incoming observation might be assigned, in principle, to any existing track. Of course, perfect knowledge of the kinematics of the targets would resolve completely these ambiguities, allowing the system to determine the only consistent matching between the observation and the tracks. This assumption is obviously unrealistic, and it also would trivialize the tracking problem. An acceptable solution may rely on a stochastic model of the kinematics that could be used to estimate the future positions of the current targets within a reasonable timeframe.

For example, in his original paper, Reid uses Kalman filters [4] to model the kinematics of the targets together with the accuracy of the sensor system, and thus constrain the space of possible hypotheses considered at each step. We also adopt the same technique, although, as we will see, our software is predisposed to work with arbitrary prediction models. In particular, predictors based on Hidden Markov Chains [1] are currently under study.

Our system is capable of dynamically adapting to heterogeneous populations of targets (motorbikes, vans, tanks, jeeps, trucks and so on) provided that a model of their kinematics is available. Such models are retrieved and uploaded dynamically (possibly from remote sources) during the processing of single observations that are typically accompanied by a "vehicle type" label. The result is an extremely flexible and powerful system expandable *ad infinitum* to incorporate different tracking logics that are able to process observations of the most diverse phenomena: mechanical targets (as in this case), malicious software propagation (e.g., worms) or people movements and itineraries.

Basics on MHT

Kalman Filters: A Kalman filter is a recursive solution to the discrete-data linear filtering problem [3, 4]. It assumes that the random process to be estimated can be modeled in the form

$$\mathbf{x_{k+1}} = \mathbf{\Phi x_k} + \mathbf{\Gamma w_k}. \tag{1}$$

The observation is assumed to occur at discrete points in time following the linear relationship

$$\mathbf{z_k} = \mathbf{H x_k} + \mathbf{v_k}, \tag{2}$$

where

$\mathbf{x_k}$ $(n \times 1)$ process state vector at time t_k,

$\mathbf{\Phi}$ $(n \times n)$ transition matrix relating $\mathbf{x_k}$ to $\mathbf{x_{k+1}}$ in the absence of a forcing function,

$\mathbf{w_k}$ $(n \times 1)$ process noise vector, assumed to be a white sequence with known covariance matrix Q,

$\mathbf{\Gamma}$ $(n \times n)$ precision matrix that selects or amplifies random kinematics effects due to the pilot,

$\mathbf{z_k}$ $(m \times 1)$ vector measurement at time t_k,

$\mathbf{H}$ $(m \times n)$ measurement matrix in absence of measurement noise,

$\mathbf{v_k}$ $(m \times 1)$ measurement error, assumed white sequence with known covariance matrix R,

$\mathbf{z_k}$ $(n \times 1)$ vector measurement at time t_k.

If the measurements could be uniquely associated with each target, then the conditional probability distribution of the state variables of each target would be a multivariate normal distribution given by the Kalman filter. The mean $\overline{x}$ and covariance $\overline{P}$ of this distribution evolve with time according to the following "time update" equations. In these equations, "overline" means predicted value, while "hat" means estimated value:

$$\mathbf{\overline{x}_{k+1}} = \mathbf{\Phi \hat{x}_k}\ , \qquad \mathbf{\overline{P}_{k+1}} = \mathbf{\Phi \hat{P}_k} + \mathbf{\Gamma Q \Gamma^T}\ . \tag{3}$$

When a measurement is received, the predicted values are combined with the measurement to obtain the estimated values. Note the use of the blending factor $\mathbf{K}$ to produce that combination:

$$\begin{aligned} \mathbf{\hat{x}_k} &= \mathbf{\overline{x}_k} + \mathbf{K}[\mathbf{z_k} - \mathbf{H\overline{x}_k}] & (4)\\ \mathbf{\hat{P}_k} &= \mathbf{\overline{P}_k} - \mathbf{\overline{P}_k H^T (H \overline{P}_k H^T + R)^{-1} H \overline{P}_k} & (5)\\ \mathbf{K} &= \mathbf{\hat{P}_k H^T R^{-1}}\ . & (6) \end{aligned}$$

Multiple Hypotheses: At any given time during the recursive algorithm, MHT keeps the hypotheses that have the highest likelihoods. Each time a new observation is received by the MHT algorithm, the algorithm must generate the set of all possible hypotheses that explain such an observation. There are three basic possibilities:

- The observation is added to an existing track. To compute the likelihood of this hypothesis, the distance between the current observation and the predicted state of the track must be computed. In general terms, the farther the observation is from the predicted value, the less likely it is that the association is correct. The distance is normalized using the covariance of the prediction error and compared to a threshold value to decide whether it is possible that the observation belongs to the track.
- The observation corresponds to a new target, in which case a new track is created.
- The observation is due to noise or clutter in the sensing system, in which case it is discarded.

Therefore, generally each hypothesis in the current hypothesis set generates several hypotheses each time a new observation is added. To avoid excessive increase in the number of hypotheses handled by the algorithm, it is common (and implemented in this system) to perform a pruning operation after adding an observation, so that only the most likely hypotheses are kept.

Computation of the likelihoods: The likelihood of each hypothesis (P_i^k) is obtained recursively from that of

the hypothesis from which it branches (P_g^{k-1}). A detailed description of the derivation of the formula for the probability is provided in [3]. This formula shows the likelihood of each hypothesis after adding a collection of simultaneous measurements (type I sensor, according to the terminology in [3]):

$$P_i^k = \frac{1}{c} P_D^{NT} (1 - P_D)^{(N_{TGT} - N_{DT})} \beta_{FT}^{N_{FT}} \beta_{NT}^{N_{NT}} \quad (7)$$

$$\cdot \left[\prod_{m=1}^{N_{DT}} N(Z_m - H\overline{x}, B) S_m \right] P_g^{k-1}, \quad (8)$$

where c is a normalizing factor, P_D is the probability of detection, N_{DT} is the number of detected or confirmed targets in the measurements (Z_m), N_{TGT} is the total number of targets implied by the previous hypothesis, N_{FT} is the number of false targets among the measurements, N_{NT} is the number of new targets, β_{FT} and β_{NT} are the densities of false targets and new targets, respectively, Z_m is the mth observation in the collection that is being added, B is the covariance of the prediction error, and S_m is the probability that the type of target in the mth observation matches that of the existing track. We added this last term S_m to account for the possibility of tracking different target types, which (according to our model) can be distinguished within a certain resolution by the sensor system (i.e., there is in general a nonzero probability that an observation labeled with a certain target type really corresponds to a different type).

If the observations are added one at a time, the same formula is valid, although in this case only one from N_{DT}, N_{NT}, N_{FT} will be 1, (and the other two will be 0), and there are no $(1 - P_D)$ terms.

MHT-Kalman Implementation Details

As mentioned before, Reid's MHT uses a Kalman Filter predictor in order to decide whether to associate a new observation with an existing track and to compute hypothesis likelihoods. Formally, a Kalman Filter can be defined as a pair of specifications: $KF =< \mathcal{K}, \mathcal{O} >$, where $\mathcal{K} =< \Phi(T), \Gamma, Q(T) >$ captures the kinematic model of the target, and $\mathcal{O} =< H, R >$ specifies the "observer" characteristics, i.e., the functionalities and accuracy of the sensor that returns observations of the target's state (see previous section). Note that Φ and Q are functions of time. In general, the tracking algorithm will need a particular instance of these specifications in order to cope with a reported target. The information is codified using advanced mark-up formatting languages like XML or DAML, and is stored in relational databases that can be remotely accessed and queried by any instance of MHT-Kalman.

In what follows, we describe some of the main Java classes that implement the core of the MHT-Kalman logic.

The `MHT` class implements our variation of Reid's MHT algorithm. The following parameters are relevant: beta values β_{NT}, β_{FT}, which are density of detected, but unlabeled, targets and the density of false targets, respectively; P_D, which is the probability of detecting an object and is part of the sensor's XML specification; ν^2, which is the threshold for adding a new observation to an existing track, specifying the maximum relative distance from the actual observation to the predicted state; and *observer*, which is an instance of the class Observer described below.

Our MHT algorithm associates a likelihood object with each track. This object contains the (recursively defined) value of the current likelihood of the hypothesis: P_i^k. The computation of P_i^{k+1} then requires the above parameters plus information provided by each track through their private Kalman filter.

Class MHT maintains a hypothesis set, which is updated each time a new observation is added. Class MHT computes the new hypotheses that branch from the existing hypotheses to explain the received observation, and prunes the results so that only the most likely ones are kept.

The `Observer` class handles the $\mathcal{O}$ specifications. When instantiated, the class accesses a database of XML/DAML specifications and extracts the matrices H and R and the detection probability P_D.

The `VehicleKinematics` class handles the $\mathcal{K}$ specifications. When instantiated, the class accesses, as before, a database of XML/DAML specifications and extracts the matrices $\Phi(T), \Gamma, andQ(T)$, as well as the maximum speed of the target (which is used as a parameter to the Kalman filter, and indeed other properties of the target kinematics could be added in future versions). Since $Q(T)$ and $\Phi(T)$ are in general functions of time, the respective entries will contain mathematical expressions involving a free variable, namely, time T. Thus, the class parses the entries of those matrices and generates a convenient representation of their expressions in the form of trees of operators, which are stored inside `MathParser` objects.

The `Predictor` class is an interface for a subclass that performs predictions using a Kalman filter. Predictor objects are particular to each track. In other words, when a new track is created, a new Kalman Filter is created and associated with that track (see Figure 1).

The `KalmanFilter` class implements the Predictor interface. It provides all the functionalities of a Kalman Filter. In particular, it requires, at the beginning, an `Observer` object, a `VehicleKinematics` object and an initial `Observation` of the target's state in order to build and initialize the filter. This implies the creation and initialization of all the needed data structures. These include:

1. $\overline{x}(t), \overline{P}(t)$: mean position and error covariance;
2. Z: state observation necessary to produce a prediction (filter update);
3. $B^{-1}, \|B\|$: auxiliary matrix useful to compute the filter value K (see Reid's paper);
4. $\Phi(T), \Gamma, Q(T), H, R$: parameters of the filter.

The state prediction depends on the Kalman parameters and on two dynamic elements: the *mean state* of the target

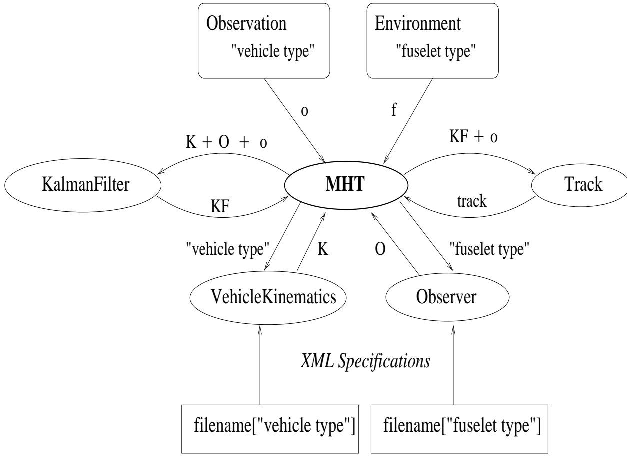

Figure 1: Dynamic Creation of a Predictor object (a Kalman filter) necessary to start a new track.

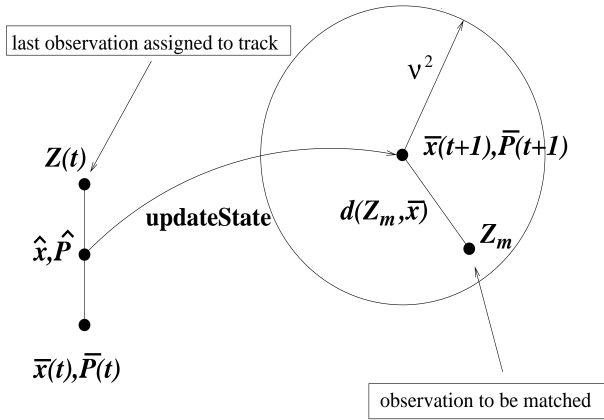

Figure 2: Use of a Kalman filter to assign observations to an existing track.

$\overline{x}$, and the *mean error covariance matrix* $\overline{P}$. These are, in general, specific to each track, and should be part of its hidden state (encapsulated in the Predictor class). Thus, each track is given a Predictor that the MHT Algorithm uses transparently to predict new states and evaluate the metric $d_B(Z_m, \overline{x}) = (Z_m - H\overline{x})^T B^{-1}(Z_m - H\overline{x})$, where Z_m is the candidate observation to be matched against the existing track being modeled by the respective Kalman filter. This value is then used to check whether the assignment criterion: $d_B(Z_m, H\overline{x}) \leq \nu^2$ is verified (see Fig. 2).

4. TRACKING ACTIVE WORMS

TRAFEN abstracts away the details of observation collection and hypothesis management, allowing developers to devote their entire attention to producing an effective, underlying model for the observed system. As a test of TRAFEN's extensibility, we applied TRAFEN to the problem of detecting active Internet worms [5].

An active Internet worm is malicious software (or malware)

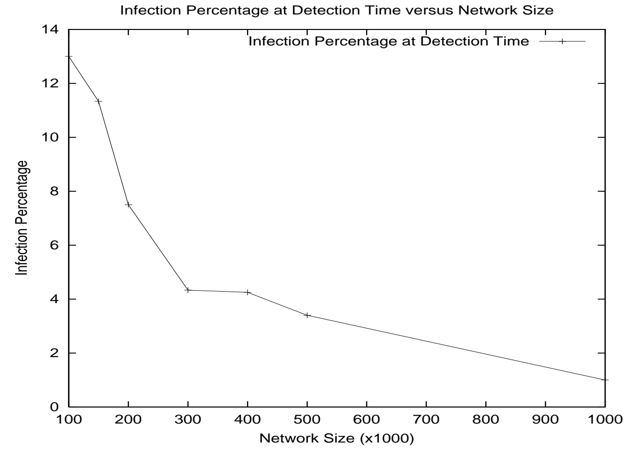

Figure 3: The percentage of vulnerable machines that are infected at the time that TRAFEN detects a simulated worm.

that autonomously travels from host to host, searching for vulnerable, uninfected systems. Recent examples include Code Red v2 and Sapphire, which exploited flaws in Microsoft servers and infected 360,000 and 75,000 machines respectively [6, 7]. Code Red, Sapphire and most other active worms find vulnerable machines by generating pseudo-random IP address and then probing if the desired vulnerable service is running at those addresses. In addition, these worms typically probe addresses as quickly as possible, so that they can infect as many machines as possible before system administrators can detect and react to them. Sapphire, for example, infected most vulnerable machines within five minutes of its launch [7], far exceeding human response time and indicating the need for *automated* worm detection.

One attractive way to detect a propagating worm automatically is by monitoring ICMP Destination Unreachable (or ICMP-T3) messages. When one host attempts to contact an unreachable target address, the last router on the path, if configured to do so, will send an ICMP-T3 message to the sending host. The last router on the path is the router that decides that the target address is unreachable; this router may be anywhere between the sender's network and the target's network. Active worms, through the process of probing randomly selected IP addresses, will attempt to contact many unreachable machines, and will generate an increasing amount of ICMP-T3 activity as they spread and infect more machines. Observing this increase is a reliable, and early, indicator of worm activity.

To detect worms based on their induced ICMP-T3 behavior, we deploy instrumented routers that send copies of any generated ICMP-T3 messages to a set of analysis stations. These analysis stations aggregate the ICMP-T3 messages, and generate a scan alert whenever a single source address tries to contact the same service on a configurable number of unreachable hosts (as well as in several other cases corresponding to different kinds of scans). These alerts,

which are expressed in XML, become the observations that are fed into TRAFEN. Since TRAFEN provides the observation- and hypothesis-management infrastructure, the only customization consists of two Java classes, one that converts the XML observation to and from an appropriate internal representation, and one that encapsulates a ruleset that determines the likelihood that an observation (scan) sequence indicates a propagating worm. TRAFEN loads both classes dynamically, and accesses them through a domain-independent interface. In our deployed prototype, the ruleset is suitable for fast-moving worms such as Code Red and Sapphire, and expresses likelihood as a function of the number of related scans occurring within different time windows.

Figure 3 shows the detection performance of this ruleset for a simulated worm. The x-axis is the number of addresses in the simulated network (in thousands), and the y-axis is the percentage of vulnerable machines that the worm infects before it is detected. In this simulation, 25% of the address space was reachable, 1% of the address space was reachable *and* vulnerable to the worm's exploit (both an unusually high percentage), and 2% of the routers were instrumented to send their ICMP-T3 messages to the analysis stations. The graph shows that detection performance increases as the size of the simulated Internet increases. With a network size of one million addresses, TRAFEN is able to detect the worm before even 1% of the vulnerable machines are infected, providing ample time to take defensive action. When deployed on the current IPv4 Internet (having 2^{32} addresses) where a 2% router coverage corresponds to approximately 3.5 Class A networks (which is achievable with cooperation from a minimal number of service providers), detection performance is expected to improve even further.

In applying TRAFEN to the worm-detection problem, we needed to provide only state representation and likelihood functions, allowing us to focus on how worms behave, rather than how to manage a large set of observations. As the worm-detection project continues, we are developing Kalman filter and Hidden Markov models that can detect stealthy and slow-moving worms. All of these models, which are concrete instantiation of process query models, can be expressed as compact classes and simply dropped into the TRAFEN infrastructure, allowing extremely rapid development. An added benefit is the ability to look at the same datastream with multiple instances of TRAFEN, each using a different process query model, to provide multiple concurrent views on worm epidemics. Realizing that the current process query model is rudimentary, yet very effective, more refined models certainly will lead to even faster and more accurate detection of active Internet worms with lower router coverages.

Work in Progress

- Type I versus Type II Sensors. When dealing with Type I Sensors, we should be able to work with sets of observations. Thus, we are extending the MHT algorithm to process objects of a class ObservationsSet that may contain only one Observation object, as in case of type II Sensors.
- Specification Ontologies. We are exploiting on-edge technology to define suitable distance metrics between kinematics and fuselet specifications. The purpose of this is to quantify and improve adaptation capabilities of the system when trying to retrieve the "best" specifications for the entities being tracked.
- Worm Detection. In addition to improving the models, we are looking at Internet-scale deployment of the ICMP-T3 sensor network, as well as developing a version that can be used within individual intranets.

References

[1] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceeding of the IEEE*, 77, Num. 2:257–286, 1989.

[2] Deborah L. McGuinness, Richard Fikes, James Hendler, and Lynn Andrea Stein. DAML+OIL: An ontology language for the semantic web. *IEEE Intelligent Systems*, 17(5), September/October 2002.

[3] Donald B. Reid. An algorithm for Tracking Multiple Targets. *IEEE Transactions on Automatic Control*, AC-24, Num. 6:843–854, 1979.

[4] Robert G. Brown and Patrick Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering.* John Wiley & Sons, 1983.

[5] Vincent H. Berk, Robert S. Gray, and George Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proceedings of AeroSense 2003: SPIE's 17th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Orlando, Florida, April 2003.

[6] David Moore, Colleen Shannon, and Jeffery Brown. Code Red: A case study on the spread and victims of an Internet worm. In *Proceedings of the Second Internet Measurement Workshop (IMW 2002)*, Marseille, France, November 2002.

[7] David Moore, Vern Paxon, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. The spread of the Sapphire/Slammer worm. CAIDA Technical Report, Cooperative Association for Internet Data Analysis (CAIDA), 2003.