DETECT: a novel framework for the detection of attacks to critical infrastructures

F. Flammini[1 2], A. Gaglione[2], N. Mazzocca[2] & C. Pragliola[1]
[1] *ANSALDO STS - Ansaldo Segnalamento Ferroviario S.p.A., Naples, Italy*
[2] *Università di Napoli "Federico II" - Dipartimento di Informatica e Sistemistica, Naples, Italy*

ABSTRACT: Critical Infrastructure Protection (CIP) against potential threats has become a major issue in modern society. CIP involves a set of multidisciplinary activities and requires the adoption of proper protection mechanisms, usually supervised by centralized monitoring systems. This paper presents the motivation, the working principles and the software architecture of DETECT (DEcision Triggering Event Composer & Tracker), a new framework aimed at the automatic and early detection of threats against critical infrastructures. The framework is based on the fact that non trivial attack scenarios are made up by a set of basic steps which have to be executed in a predictable sequence (with possible variants). Such scenarios are identified during Vulnerability Assessment which is a fundamental phase of the Risk Analysis for critical infrastructures. DETECT operates by performing a model-based logical, spatial and temporal correlation of basic events detected by the sensorial subsystem (possibly including intelligent video-surveillance, wireless sensor networks, etc.). In order to achieve this aim, DETECT is based on a detection engine which is able to reason about heterogeneous data, implementing a centralized application of "data fusion". The framework can be interfaced with or integrated in existing monitoring systems as a decision support tool or even to automatically trigger adequate countermeasures.

1 INTRODUCTION & BACKGROUND

Critical Infrastructure Protection (CIP) against terrorism and any form or criminality has become a major issue in modern society. CIP involves a set of multidisciplinary activities, including Risk Assessment and Management, together with the adoption of proper protection mechanisms, usually supervised by specifically designed Security Management Systems (SMS)[1] (see e.g. (LENEL 2008)).

Among the best ways to prevent attacks and disruptions is to stop any perpetrators before they strike. This paper presents the motivation, the working principles and the software architecture of DETECT (DEcision Triggering Event Composer & Tracker), a new framework aimed at the automatic detection of threats against critical infrastructures, possibly before they evolve to disastrous consequences. In fact, non trivial attack scenarios are made up by a set of basic steps which have to be executed in a predictable sequence (with possible variants). Such scenarios must be precisely identified during Vulnerability Assessment which is a fundamental aspect of Risk Analysis for critical infrastructures (Lewis 2006). DETECT operates by performing a model-based logical, spatial and temporal correlation of basic events detected by intelligent video-surveillance and/or sensor networks, in order to "sniff" sequence of events which indicate (as early as possible) the likelihood of threats. In order to achieve this aim, DETECT is based on a detection engine which is able to reason about heterogeneous data, implementing a centralized application of "data fusion" (a well-known concept in the research field of cognitive / intelligent autonomous systems (Tzafestas 1999)). The framework can be interfaced with or integrated in existing SMS/SCADA systems in order to automatically trigger adequate countermeasures.

With respect to traditional approaches of infrastructure surveillance, DETECT allows for:

- A quick, focused and fully automatic response to emergencies, possibly independent from human supervision and intervention (though manual confirmation of detected alarms remains an option). In fact, human management of critical situations, possibly involving many simultaneous

[1] In some cases, they are integrated in the traditional SCADA (*Supervisory Control & Data Acquisition*) systems.

events, is a very delicate task, which can be error prone as well as subject to forced inhibition.

- An early warning of complex attack scenarios since their first evolution steps using the knowledge base provided by experts during the qualitative risk analysis process. This allows for preventive reactions which are very unlikely to be performed by human operators given the limitation both in their knowledge base and vigilance level. Therefore, a greater situational awareness can be achieved.
- An increase in the Probability Of Detection (POD) while minimizing the False Alarm Rate (FAR), due to the possibility of logic as well as temporal correlation of events. While some SMS/SCADA software offer basic forms of logic correlation of alarms, the temporal correlation is not implemented in any nowadays systems, to the best of our knowledge (though some vendors provide basic options of on-site configurable "sequence" correlation embedded in their multi-technology sensors).

The output of DETECT consists of:

- The identifier(s) of the detected/suspected scenario(s).
- An alarm level, associated to scenario evolution (only used in deterministic detection as a linear progress indicator; otherwise, it can be set to 100%).
- A likelihood of attack, expressed in terms of probability (only used as a threshold in heuristic detection; otherwise, it can be set to 100%).

DETECT can be used as an on-line decision support system, by alerting in advance SMS operators about the likelihood and nature of the threat, as well as an autonomous reasoning engine, by automatically activating responsive actions, including audio and visual alarms, emergency calls to first responders, air conditioning flow inversion, activation of sprinkles, etc.

The main application domain of DETECT is homeland security, but its architecture is suited to other application fields like environmental monitoring and control, as well. The framework is being experimented in railway transportation systems, which have been demonstrated by the recent terrorist strikes to be among the most attractive and vulnerable targets. Example attack scenarios include intrusion and drop of explosive in subway tunnels, spread of chemical or radiological material in underground stations, combined attacks with simultaneous multiple train halting and railway bridge bombing, etc. DETECT has proven to be particularly suited for the detection of such articulated scenarios using a modern SMS infrastructure based on an extended network of cameras and sensing devices. With regards to the underlying security infrastructure, a set of interesting technological and research issues can also be addressed, ranging from object tracking algorithms to wireless sensor network integration; however, these aspects (mainly application specific) are not in the scope of this work.

DETECT is a collaborative project carried out by the Business Innovation Unit of Ansaldo STS Italy and the Department of Computer and System Science of the University of Naples "Federico II".

The paper is organized as follows: Section 2 presents a brief summary of related works; Section 3 introduces the reference software architecture of the framework; Section 4 presents the language used to describe the composite events; Section 5 describes the implementation of the model-based detection engine; Section 6 contains a simple case-study application; Section 7 draws conclusions and provides some hints about future developments.

2 RELATED WORKS

Composite event detection plays an important role in the active database research community, which has long been investigating the application of Event Condition Action (ECA) paradigm in the context of using triggers, generally associated with update, insert or delete operations. In HiPAC (Dayal et al. 1988) active database project an event algebra was firstly defined.

Our approach for composite event detection follows the semantics of the Snoop (Chakravarthy & Mishra 1994) event algebra. Snoop has been developed at the University of Florida and its concepts have been implemented in a prototype called Sentinel (Chakravarthy et al. 1994, Krishnaprasad 1994). Event trees are used for each composite event and these are merged to form an event graph for detecting a set of composite events. An important aspect of this work lies in the notion of *parameter contexts*, which augment the semantics of composite events for computing their parameters (parameters indicate "component events"). CEDMOS (Cassandra et al. 1999) refers to the Snoop model in order to encompass heterogeneity problems which often appear under the heading of sensor fusion. In (Alferes & Tagni 2006) the implementation of an event detection engine that detects composite events specified by expressions of an illustrative sublanguage of the Snoop event algebra is presented. The engine ha been implemented as a Web Service, so it can also be

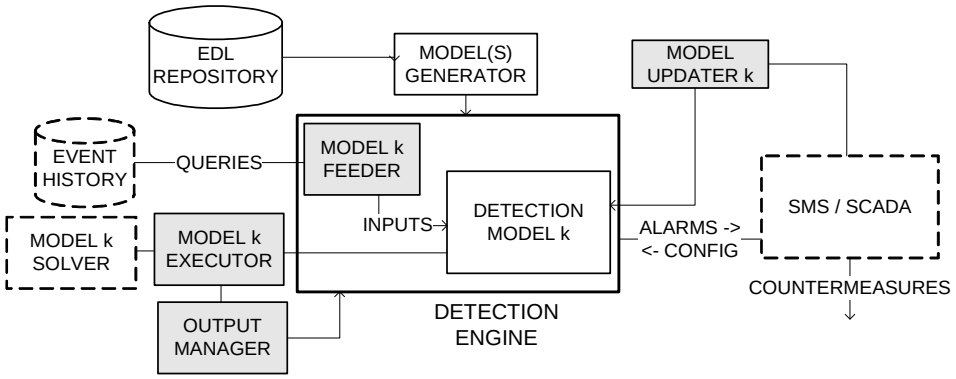

Figure 1. The software architecture of DETECT

used by other services and frameworks if the markup for the communication of results is respected.

Different approaches for composite event detection are taken in Ode (Gehani et al. 1992a, b) and Samos (Gatziu et al. 1994, Gatziu et al. 2003). Ode uses an extended Finite Automata for composite event detection while Samos defines a mechanism based on Petri Nets for modeling and detection of composite events for an Object Oriented Data-Base Management System (OODBMS).

DETECT transfers to the physical security the concept of Intrusion Detection System (IDS) which is nowadays widespread in computer (or "logical") security, also borrowing the principles of Anomaly Detection, which is applied when an attack pattern is known a priori, and Misuse Detection, indicating the possibility of detecting unknown attacks by observing a significant statistical deviation from the normality (Jones & Sielken 2000). The latter aspect is strictly related to the field of Artificial Intelligence and related classification methods.

Intelligent video-surveillance exploits Artificial Vision algorithms in order to automatically track object movements in the scene, detecting several type of events, including virtual line crossing, unattended objects, aggressions, etc. (Remagnino et al. 2007).

Sensing devices include microwave / infrared / ultrasound volumetric detectors/barriers, magnetic detectors, vibration detectors, explosive detectors, and advanced Nuclear Bacteriologic Chemical Radiological (NBCR) sensors (Garcia 2001). They can be connected using both wired and wireless networks, including ad-hoc Wireless Sensor Networks (WSN) (Lewis 2004, Roman et al. 2007).

3 THE SOFTWARE ARCHITECTURE

The framework is made up by the following main modules (see Figure 1):

- **Event History** database, containing the list of basic events detected by sensors or cameras, tagged with a set of relevant attributes including detection time, event type, sensor id, sensor type, sensor group, object id, etc. (some of which can be optional, e.g. "object id" is only needed when video-surveillance supports inter-camera object tracking).
- **Attack Scenario Repository**, providing a database of known attack scenarios as predicted in Risk Analysis sessions and expressed by means of an *Event Description Language* (EDL) including logical as well as temporal operators (derived from (Chakravarthy et al. 1994)).
- **Detection Engine**, supporting both deterministic (e.g. *Event Trees*, *Event Graphs*) and heuristic (e.g. *Artificial Neural Networks*, *Bayesian Networks*) models, sharing the primary requirement of real-time solvability (which excludes e.g. *Petri Nets* from the list of candidate formalisms).
- **Model Generator**, which has the aim of building the detection model(s) (structure and parameters) starting from the Attack Scenario Repository by parsing all the EDL files.
- **Model Manager**, constituted by four sub-modules:
  - **Model Feeder** (one for each model), which instantiates the inputs of the detection engine according to the nature of the models by cyclically performing proper queries and data filtering on the Event History (e.g. selecting sensor typologies and zones, excluding temporally distant events, etc.).
  - **Model Executor** (one for each model), which triggers the execution of the model, once it has been instantiated, by activating the related (external) solver. An execution is usually needed at each new event detection.
  - **Model Updater** (one for each model), which is used for on-line modification of the model (e.g. update of a threshold parameter), without

regenerating the whole model (whenever supported by the modeling formalism).

- **Output Manager** (single), which stores the output of the model(s) and/or passes it to the interface modules.

- **Model Solver**, that is the existing or specifically developed tool used to execute the model.

Model Generator and Model Manager are dependent on the formalisms used to express the models constituting the Detection Engine. In particular, the Model Generator and Model Feeder are synergic in implementing the detection of the event specified in EDL files: in fact, while the Detection Engine plays undoubtedly a central role in the framework, many important aspects are demanded to the way the query on the database is performed (i.e. selection of proper events). As an example, in case the Detection Engine is based on Event Trees (a combinatorial formalism), the Model Feeder should be able to pick the set of last N consecutive events fulfilling some temporal properties (e.g. total time elapsed since the first event of the sequence < T), as defined in the EDL file. In case of Event Graphs (a state-based formalism), instead, the model must be fed by a single event at a time.

Besides these main modules, there are others which are also needed to complete the framework with useful, though not always essential, features (some of which can also be implemented by external tools or in the SMS):

- **Scenario GUI** (*Graphical User Interface*) used to draw attack scenarios using an intuitive formalism and a user-friendly interface (e.g. specifically tagged UML Sequence Diagrams stored in the standard XMI[2] format (Object Management Group UML 2008)).

- **EDL File Generator**, translating GUI output into EDL files.

- **Event Log**, in which storing information about composite events, including detection time, scenario type, alarm level and likelihood of attack (whenever applicable).

- **Countermeasure Repository**, associating to each detected event or event class a set of operations to be automatically performed by the SMS.

- Specific **drivers** and **adapters** needed to interface external software modules, possibly including anti-intrusion and video-surveillance subsystems.

- Standard **communication protocols** (OPC[3], ODBC[4], Web-Services, etc.) needed to interoperate with open databases, SMS/SCADA, or any other client/server security subsystems which are compliant to such standards.

The last two points are necessary to provide DETECT with an open, customizable and easily upgradeable architecture. For instance, by adopting a standard communication protocol like OPC, an existing SMS supporting this protocol could integrate DETECT as it was just a further sensing device.

At the current development state of DETECT:

- A GUI has been developed to edit scenarios and generate EDL files starting from the Event Tree graphical formalism.

- A Detection Engine based on Event Graphs (Buss 1996) is already available and fully working, using a specifically developed Model Solver.

- A Model Generator has been developed in order to generate Event Graphs starting from the EDL files in the Scenario Repository.

- A Web Services based interface has been developed to interoperate with external SMS.

- The issues related to the use of ANN (Jain et al. 1996) for heuristic detection have been addressed and the related modules are under development and experimentation.

4 THE EVENT DESCRIPTION LANGUAGE

The Detection Engine needs to recognize combination of events, bound each other with appropriate operators in order to form composite events of any complexity. Generally speaking, an event is a happening that occurs in the system, at some location and at some point in time. In our context, events are related to sensor data variables (i.e. variable *x* greater than a fixed threshold, variable *y* in a fixed range, etc.). Events are classified as *primitive events* and *composite events.*

A primitive event is a condition on a specific sensor which is associated some parameters (i.e. event identifier, time of occurrence, etc). Event parameters can be used in the evaluation of conditions. Each entry stored in the Event History is a quadruple:

< IDev, IDs, IDg, tp >, where:

[2] XML (*eXtended Markup Language)* Metadata Interchange.

[3] OLE (*Object Linking & Embedding*) for Process Communication.

[4] Open Data-Base Connectivity.

- **IDev** is the event identifier;
- **IDs** is the sensor identifier;
- **IDg** is the sensor group identifier (needed for geographical correlation);
- **tp** is the event occurrence time which should be a sensor timestamp (when a global clock is available for synchronization) or the Event History machine clock.

Since the message transportation time is not instantaneous, the event occurrence time can be different from the registration time. Several research works have addressed the issue of clock synchronization in distributed systems. Here we assume that a proper solution (e.g. time shifting) has been adopted at a lower level.

A composite event is a combination of primitive events defined by means of proper operators. The EDL of DETECT is derived from Snoop event algebra (Chakravarthy & Mishra 1994). Every composite event instance is a triple:

< IDec, parcont, te >, where:

- **IDec** is the composite event identifier;
- **parcont** is the parameter context, stating which occurrences of primitive events need to be considered during the composite event detection (as described below);
- **te** is the temporal value related to the occurrence of the composite event (corresponding to the tp of the last component event).

Formally an event E (either primitive or composite) is a function from the time domain onto the boolean values, *True* and *False*:

$$E: T \rightarrow \{\text{True, False}\}, \text{ given by:}$$

$$E(t) = \begin{cases} \textit{True}, \text{ if E occurs at time t} \\ \textit{False}, \text{ otherwise} \end{cases}$$

The basic assumption of considering a boolean function is quite general, since different events can be associated to a continuous sensor output according to a set of specified thresholds. Furthermore, negate conditions (!E) can be used when there is the need for checking that an event is no longer occurring. This allows considering both instantaneous ("occurs" = "has occurred") and continuous ("occurs" = "is occurring") events. However, in order to simplify EDL syntax, negate conditions on events can be substituted by

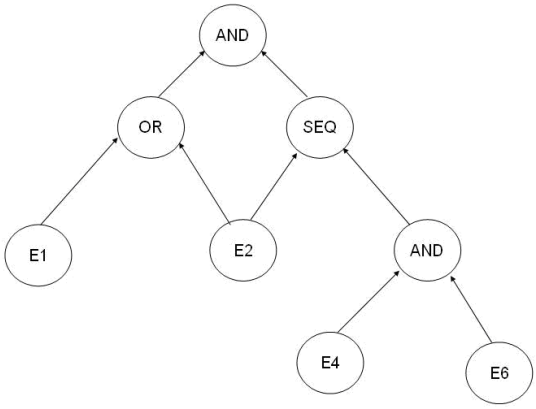

Figure 2. Event tree for composite event ((E1 OR E2) AND (E2 SEQ (E4 AND E6)))

complementary events. An event E_c is complementary to E when:

$$E_c \Rightarrow !E$$

Each event is denoted by an *event expression*, whose complexity grows with the number of involved events. Given the expressions E_1, E_2, …, E_n, every application on them through any operator is still an expression. In the following, we briefly describe the semantics of these operators. For a formal specification of these semantics, the reader can refer to (Chakravarthy et al. 1994).

OR. Disjunction of two events E_1 and E_2, denoted (E_1 OR E_2). It occurs when at least one of its components occurs.

AND. Conjunction of two events E_1 and E_2, denoted (E_1 AND E_2). It occurs when both E_1 and E_2 occur (the temporal sequence is ignored).

ANY. A composite event, denoted ANY (m, E_1, E_1, …, E_n), where $m \leq n$. It occurs when m out of n distinct events specified in the expression occur (the temporal sequence is ignored).

SEQ. Sequence of two events E_1 and E_2, denoted (E_1 SEQ E_2). It occurs when E_2 occurs provided that E_1 has already occurred. This means that the time of occurrence of E_1 has to be less than the time of occurrence of E_2.

The sequence operator is used to define composite events when the order of its component events is relevant. Another way to perform a time correlation on events is by exploiting temporal constraints.

The logic correlation could loose meaningfulness when the time interval between component events exceeds a certain threshold. *Temporal constraints* can be defined on primitive events with the aim of defining a validity interval for the composite event. Such constraints can be added to any operator in the formal expression used for event description.

For instance, let us assume that in the composite event E = (E_1 AND E_2) the time interval between the occurrence of primitive events E_1 and E_2 must be at

most T. The formal expression is modified by adding the temporal constraint [T] as follows:

$$(E_1 \text{ AND } E_2) [T] = \textit{True}$$
$$\Leftrightarrow$$
$$\exists\, t_1 < t \mid (E_1(t) \wedge E_2(t_1) \vee E_1(t_1) \wedge E_2(t)) \wedge |t - t_1| \leq T$$

5 THE SOFTWARE IMPLEMENTATION

This section describes some implementation details of DETECT, referring to the current development state of the core modules of the framework, including the Detection Engine. The modules have been fully implemented using the Java programming language. *JGraph* has been employed for the graphical construction of the Event Trees used in the Scenario GUI. Algorithms have been developed for detecting composite events in all parameter contexts.

Attack scenarios are currently described by Event Trees, where leaves represent primitive events while internal nodes (including the root) represent EDL language operators. Figure 2 shows an example Event Tree representing a composite event.

After the user has sketched the Event Tree, the Scenario GUI module parses the graph and provides the EDL expression to be added to the EDL Repository. The parsing process starts from the leaf nodes representing the primitive events and ends at the root node. Starting from the content of the EDL Repository, the Model Generator module builds and instantiates as many *Event Detector* objects as many composite events stored in the database. The detection algorithm implemented by such objects is based on Event Graphs and the objects include the functionalities of both the Model Solver and the Detection Engine.

In the current prototype, after the insertion of attack scenarios, the user can start the detection process on the Event History using a stub front-end (simulating the Model Executor and the Output Manager modules). A primitive event is accessed from the database by a specific Model Feeder module, implemented by a single *Event Dispatcher* object which sends primitive event instances to all Event Detectors responsible for the detection process.

The Event Dispatcher requires considering only some event occurrences, depending on a specific policy defined by the *parameter context*. The policy is used to define which events represent the beginning (*initiator*) and the end (*terminator*) of the scenario. The parameter context states which component event occurrences play an active part in the detection process. Four contexts for event detection can be defined:

- *Recent*: only the most recent occurrence of the initiator is considered.
- *Chronicle:* the (initiator, terminator) pair is unique. The oldest initiator is paired with the oldest terminator.
- *Continuous:* each initiator starts the detection of the event.
- *Cumulative*: all occurrences of primitive events are accumulated until the composite event is detected.

The effect of EDL operators is then conditioned by the specific context, which is implemented in the Event Dispatcher. Theoretically, in the construction of the model a different node should be defined for each context. Whilst a context could be associated to each operator, currently a single context is associated to each detection model. Furthermore, a different node object for each context has been implemented.

In the current implementation, Event Graphs are used to detect the scenarios defined by Event Trees, which are only used as a descriptive formalism. In fact, scenarios represented by more Event Trees can be detected by a single Event Graph produced by the Model Generator. When an Event Detector receives a message indicating that an instance of a primitive event E_i has occurred, it stores the information in the node associated with E_i. The detection of composite events follows a bottom-up process that starts from primitive event instances and flows up to the root node. So the composite event is detected when the condition related to the root node operator is verified. The propagation of the events is determined by the user specified context. After the detection of a composite event, an object of a special class (*Event Detected*) is instantiated with its relevant information (identifier, context, component event occurrences, initiator, terminator).

6 AN EXAMPLE SCENARIO

In this section we provide an application of DETECT to the case-study of a subway station. We consider a composite event corresponding to a terrorist threat. The classification of attack scenarios is performed by security risk analysts in the vulnerability assessment process.

The attack scenario consists of an intrusion and drop of an explosive device in a subway tunnel. Let us suppose that the dynamic of the scenario follows the steps reported below:

1. The attacker stays on the platform for the time needed to prepare the attack, missing one or more trains.

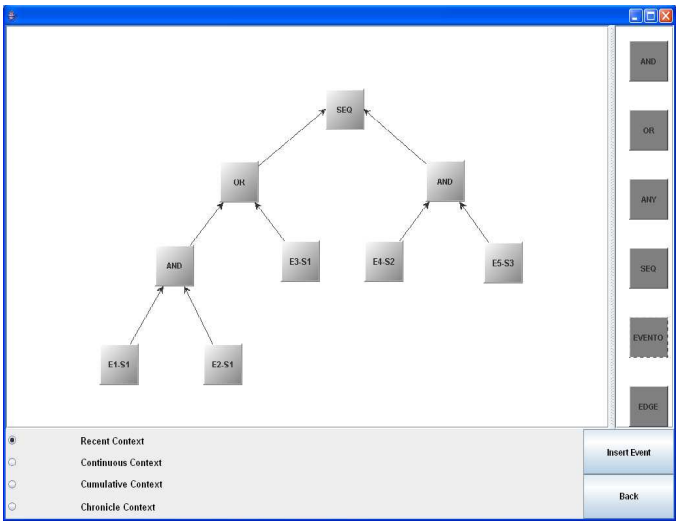

Figure 3. Insertion of the composite event using the GUI

2. The attacker goes down the tracks by crossing the limit of the platform and moves inside the tunnel portal.
3. The attacker drops the bag containing the explosive device inside the tunnel and leaves the station.

Obviously, it is possible to think of several variants of this scenario. For instance, only one between step 1 and step 2 could happen. Please note that the detection of step 1 (person not taking the train) would be very difficult to detect by a human operator in a crowded station due to the people going on and off the train.

Le us suppose that the station is equipped with a security system including intelligent cameras (S_1), active infrared barriers (S_2) and explosive sniffers (S_3) for tunnel portal protection. The formal description of the attack scenario consists of a sequence of events which should be detected by the appropriate sensors and combined in order to form the composite event.

The formal specification of primitive events constituting the scenario is provided in following:

a) extended presence on the platform (E_1 by S_1);

b) train passing (E_2 by S_1);

c) platform line crossing (E_3 by S_1);

d) tunnel intrusion (E_4 by S_2);

e) explosive detection (E_5 by S_3).

For the sake of brevity, further steps are omitted.

The composite event **drop of explosive in tunnel** can be specified in EDL as follows:

$$\mathbf{(E_1\ AND\ E_2)\ OR\ E_3\ SEQ\ (E_4\ AND\ E_5)}$$

Figure 3 provides a GUI screenshot showing the Event Tree for the composite event specified above. The user chooses the parameter context and builds the tree (including primitive events, operators and interconnection edges) by the user-friendly interface. If a node represents a primitive event, the user has to specify event (E_x) and sensor (S_x) identifiers. If a node is an operator, the user can optionally specify other parameters such as a temporal constraint, the partial alarm level and the *m* parameter (ANY operator). Also, the user can activate / deactivate the composite events stored in the repository carrying out the detection process.

A partial alarm can be associated to the scenario evolution after step 1 (left AND in the EDL expression), in order to warn the operator of a suspect abnormal behavior.

In order to activate the detection process, a simulated Event History has been created ad-hoc. An on-line integration with a real working SMS will be performed in the near future for experimentation purposes.

7 CONCLUSIONS & FUTURE WORKS

In this paper we have introduced the working principles and the software architecture of DETECT, an expert system allowing for early warnings in security critical domains.

DETECT can be used as a module of a more complex hierarchical system, possibly involving several infrastructures. In fact, most critical infrastructures are organized in a multi-level fashion: local sites, grouped into regions and then monitored centrally by a national control room, where all the (aggregated) events coming from lower levels are routed. When the entire system is available, each site at each level can benefit from the knowledge of significant events happening in other sites. When some communication links are unavailable, it is still possible to activate countermeasures basing on the local knowledge.

We are evaluating the possibility of using a single automatically trained multi-layered ANN to complement deterministic detection by: 1) classification of suspect scenarios, with a low FAR; 2) automatic detection of abnormal behaviors, by observing deviations from normality; 3) on-line update of knowledge triggered by the user when a new anomaly has been detected. The ANN model can be trained to understand normality by observing the normal use of the infrastructure, possibly for long periods of time. The Model Feeder for ANN operates in a way which is similar to the Event Tree example provided above. A ANN specific Model Updater allows for on-line learning facility. Future developments will be aimed at a more cohesive integration between deterministic and heuristic detection, by making the models interact one with

each other.

REFERENCES

Buss, A.H. 1996. Modeling with Event Graphs. In *Proc. Winter Simulation Conference*, pp. 153-160.

Cassandra, A.R.; Baker, D. & Rashid, M. 1999. CEDMOS: Complex Event Detection and Monitoring System. *MCC Tecnical Report CEDMOS-002-99*, MCC, Austin, TX.

Chakravarthy, S. & Mishra, D. 1994. Snoop: An expressive event specification language for active databases. *Data Knowl. Eng.*, Vol. 14, No. 1, pp. 1–26.

Chakravarthy, S.; Krishnaprasad, V.; Anwar, E. & Kim, S. 1994. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proceedings of the 20th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers, San Francisco, CA, pp. 606-617.

Dayal, U.; Blaustein, B.T.; Buchmann, A.P.; Chakravarthy, S.; Hsu, M.; Ledin, R.; McCarthy, D.R.; Rosenthal, A.; Sarin, S.K.; Carey, M.J.; Livny, M. & Jauhari, R. 1988. The HiPAC Project: Combining Active Databases and Timing Constraints. *SIGMOD Record*, Vol. 17, No. 1, pp. 51-70.

Garcia, M.L. 2001. *The Design and Evaluation of Physical Protection Systems*. Butterworth-Heinemann, USA.

Gatziu, S. & Dittrich, K.R. 1994. Detecting Composite Events in Active Databases Using Petri Nets. *In Proc. of the 4th International Workshop on Research Issues in data Engineering: Active Database Systems*, pp. 2-9.

Gatziu, S. & Dittrich, K.R. 2003. Events in an Object-Oriented Database System. *In Proc. of the 1st International Conference on Rules in Database Systems*, pp. 23-39.

Gehani, N.H.; Jagadish, H.V. & Shmueli, O. 1992a. Event Specification in an Object-Oriented Database. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 81-90.

Gehani, N.H.; Jagadish, H.V. & Shmueli, O. 1992b. COMPOSE A System For Composite Event Specification and Detection. *Technical report, AT&T Bell Laboratories*, Murray Hill, NJ.

J.J. Alferes, G & Tagni, E. 2006. Implementation of a Complex Event Engine for the Web. *In Proc. of IEEE Services Computing Workshops (SCW 2006)*. September 18-22. Chicago, Illinois, USA.

Jain, A.K.; Mao, J. & Mohiuddin, K.M. 1996. Artificial Neural Networks: A tutorial. In *IEEE Computer*, Vol. 29, No. 3, pp. 56-63.

Jones, A.K. & Sielken, R.S. 2000. Computer System Intrusion Detection: A Survey. *Technical Report, Computer Science Dept., University of Virginia.*

Krishnaprasad, V. 1994. Event Detection for Supporting Active Capability in an OODBMS: Semantics, Architecture and Implementation. *Master's Thesis*. University of Florida.

LENEL OnGuard 2008. http://www.lenel.com.

Lewis, F.L. 2004. Wireless Sensor Networks. In *Smart Environments: Technologies, Protocols, and Applications*, ed. D.J. Cook and S.K. Das. John Wiley, New York.

Lewis, T.G. 2006. *Critical Infrastructure Protection in Homeland Security: Defending a Networked Nation*. John Wiley, New York.

Object Management Group UML, 2008. http://www.omg.org/uml.

OLE for Process Communication. http://www.opc.org.

Remagnino, P.; Velastinm S. A.; Foresti G.L. & Trivedi M. 2007. Novel concepts and challenges for the next generation of video surveillance systems. *Machine Vision and Applications* (Springer), Vol. 18, Issue 3-4, pp. 135-137.

Roman, R.; Alcaraz, C. & Lopez, J. 2007. The role of Wireless Sensor Networks in the area of Critical Information Infrastructure Protection. *Information Security Technical Report*, Vol. 12, Issue 1, pp. 24-31.

Tzafestas, S.G. 1999. *Advances in Intelligent Autonomous Systems*. Kluwer.