

# Detecting Intrusions in Agent System by Means of Exception Handling

Eric Platon<sup>1,3</sup>, Martin Rehak<sup>2</sup>, Nicolas Sabouret<sup>3</sup>, Michal Pechoucek<sup>2</sup>,  
and Shinichi Honiden<sup>1</sup>

<sup>1</sup> National Institute of Informatics  
Sokendai and University of Tokyo  
2-1-2 Hitotsubashi, 101-8430 Tokyo

<sup>2</sup> Gerstner Laboratory  
Czech Technical University  
Technická 2, Prague, 166 27

<sup>3</sup> Laboratoire d'informatique de Paris 6  
Université Pierre et Marie Curie

104, Avenue du Président Kennedy, 75016 Paris

{platon,honiden}@nii.ac.jp, {mrehak,pechouc}@labe.felk.cvut.cz,  
Nicolas.Sabouret@lip6.fr

**Abstract.** We present a formal approach to conception of a dedicated security infrastructure based on the exception handling in the protected agents. Security-related exceptions are identified and handled by a dedicated reflective layer of the protected agent, or delegated to specialized intrusion management agents in the system if the local reflective layer fails to address the problem. Incidents are handled either directly, if a known remedy exists or indirectly, when an appropriate solution must be identified before response execution. The cooperation between the intrusion management agents and aggregation of their observations can make the system more resilient to misclassification than a solution based purely on signature matching.

## 1 Introduction

Between their other advantages, the multi-agent systems aim to achieve robust and reliable failover behavior by openness, runtime reconfiguration, dynamic replanning and partner selection and higher agent autonomy in general. Therefore, their application can significantly improve the reliability of the system when it encounters random failures, e.g. hardware malfunction, communication problems, failed battery/power sources and others [1].

On the downside, the very same features that make multi-agent systems resilient against random failures can be leveraged by the adversary to intentionally and efficiently harm the system. The administration of multi-agent systems is notoriously difficult, and when we impose restrictive policies that make the system more resilient against intentional, external intrusions, we restrict its capability to cope with random failures. Therefore, we seek an alternative to such restrictive approach, by using the same features that make the system resilient to natural faults and leveraging them for intrusion observation, detection and response.

The aim of this paper is to present a new approach to intrusion management based on a model of intrusion detection as exception, the collaboration of *self-protected agents* (or protection-aware agents) running on network hosts, and dedicated network-based Intrusion Detection System (IDS). We argue that the collaboration is crucial for autonomous and timely reactions to intrusions with low level of errors and acceptable performance. While we argue that the integration with a wide range of protected (and potentially vulnerable) agents is crucial, we do not impose any special restrictions on the type of alerts to provide to IDS, making the integration effort easier to standardize and implement.

We propose a distributed, layered architecture where the deployed network elements employ their autonomic properties tailored to manage external attacks, based on the exception-generated feedback from the protected agents. The contribution of the paper is to improve network intrusion management schemes by introducing an overlay of IDS agents, referred to as **CIME** (Collaborative Intrusion Management Element), on a network of autonomic, **protection-aware** agents. The protected agents have a reflective architecture allowing them to encompass legacy software and they are able to detect intrusions with an appropriate exception handling model.

The next section (2) presents a model of exception adapted to the detection of intrusion in the cases we are focusing on. Section 3 describes the reflective architecture of in-network agents and the structure of CIME agents. The collaboration scheme and protocols followed by protected agents and CIME on intrusion detection are detailed in section 4. Section 6 concludes the paper and presents our current and future work.

## 2 Model of Intrusion Detection as Exception

In the scope of this work, we will use the term *network intrusion* to denote situations when an adversary (human, software, or mixed groups) external to the multi-agent system has accessed a restricted data or functionality of the system or has disrupted the system use for legitimate users. Intrusion prevention requires appropriate monitoring of the protected system. Low-level monitoring of the communication network searches for irregularities, such as specific intrusion signatures or unusual traffic. On a conceptually higher level, we can model the intrusions as *exceptions* [2] in the behavior of agents in the system. An exception may trigger a collaborative exception handling mechanism that detects and possibly manages the intrusion.

### 2.1 Model

The exception model is related to the usual notion of exception in programming languages [3,4], as the aim is to provide a fault tolerance mechanism, where ‘faults’ to recover from are network intrusions instead of software or hardware problems. The semantics of the exception model differs consequently as the software involved in the exception handling has not encountered any programming issue, but it still has to handle the (potential) symptoms caused by an intrusion.

The state-chart on Fig. 1 describes the exception model in the system, whose different activities will be allocated to either protected agents or CIME in the system. The

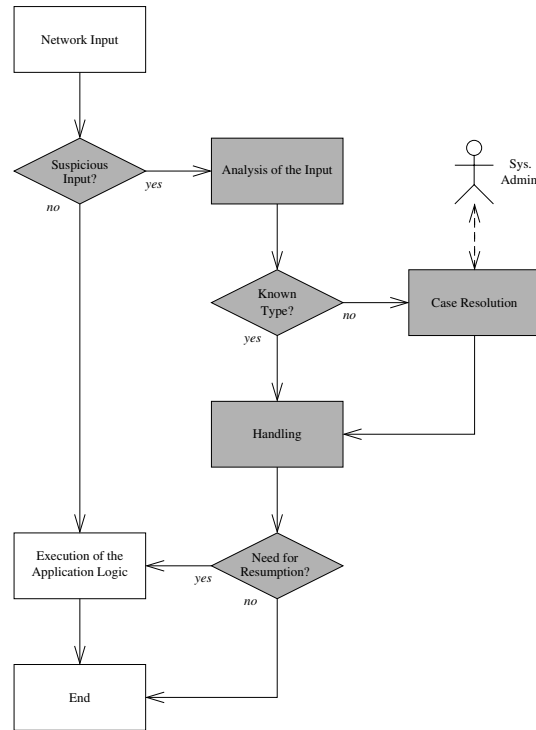


Fig. 1. State-chart for the intrusion in-network management process

state-chart defines the behavior of network elements (either network devices, specialized agents or agent platform security services) when they receive inputs from outside the system. The state-chart features two colors, namely white and gray, to describe respectively the fundamental functionalities and the exception handling stages on intrusion detection.

When an element receives a message, it first screens it to detect the *suspicious* ones, according to standard criteria (e.g. signature) and to additional ones generated by the system or introduced by administrators. If an input is not considered suspicious, it is processed in the application logic of the element, and the state-chart ends on completion of this processing. Any input deemed suspicious is further analyzed to determine the way to handle it. If it is a known intrusion, the element encompasses a handling mechanism and executes it. After completion of the handler, the input can be either processed or discarded, and the state-chart ends. Finally, if the type of intrusion is unknown, the agent attempts to resolve the case by collaborating with other elements and, if needed, with system administrators. The case resolution attempts to automatically find a remedy to the situation, with human decision in the last resort depending on degree of autonomy of the system. The result of the resolution is a command for handling, optionally followed by the resumption of the application logic, and the completion of the state-chart. The system administrator can alternatively initiate the state-chart by

sending a command to resolve specific cases. This situation occurs whenever the administrator initiates manual maintenance operations that are preventive or might be out of the scope of the system autonomic capabilities. A typical case is to deploy a patch in the system. The elements then handle the command in the sense that they update automatically their capabilities.

The state-chart shows an overview of the complete processing of inputs and commands from administrators. Two types of network elements are involved in this process, namely the *protected agents* and the *CIME*. These two types are logical and one platform is likely to host both types of elements. Agents and CIME deal with the different stages of the state-chart according to *interaction protocols*. Elements and protocols are introduced in the remainder of the paper.

### 3 Architectures of Protected Agents and CIME

#### 3.1 Architecture of Self-protected Agents

The architecture of the protected agent is designed as an overlay on a generic application agent. The architecture depicted on Fig. 2 represents an agent with the reflective extension that makes the overall element autonomic. The complete architecture forms a *self-protected agent*.

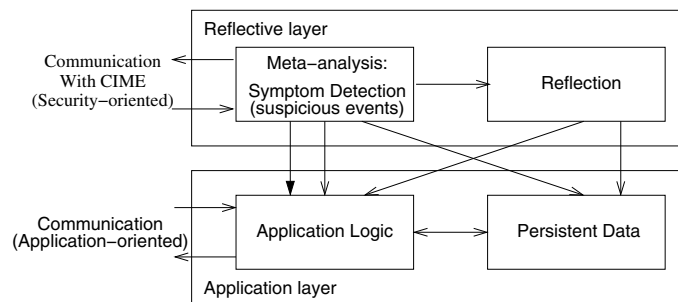


Fig. 2. Architecture of self-protected agents in the network

The architecture has two threads of executions, each in a different layer of the agent. The application layer encompasses the *application logic* (e.g. a legacy application) and its *persistent local data*. Application logic and persistent data components interact to fulfill the functions of the agent, including replying to requests received through the communication link. The reflective layer [5] encompasses two components for a second thread of execution. The *meta-analysis* refers to mechanisms that aims to detect the intrusions, signal the exceptional situations to other components and protect the agent against suspicious input (thus the 'self-protection'). It corresponds to most stages of suspicion evaluation process in Fig. 1. The *reflective* component contains the mechanisms that actually handles the suspected threats and oversees the modification of the agent's code. The meta-analysis commands the reflective mechanisms to achieve the necessary

reactions against attacks (unidirectional link). It observes the application layer components to detect suspicious inputs, and it overrides the application logic component (bold arrow) to prevent any execution of suspicious data while it is analyzed. The reflection component receives commands from the meta-analysis to apply a variety of mechanisms on the application logic and the persistent data, such as patching the agent or performing roll-backs on a database. The meta-analysis has additional communication capabilities to contact CIME components in the system. When the meta-analysis encounters a new situation that it cannot handle or decides to inform other elements in a collaboration, it contacts CIME according to an interaction protocol introduced in section 4.

The architecture treats an intrusion detection even as an exception in programming languages. It has some differences however (see Section 2), notably the termination and resumption models. The architecture allows the application layer to continue executing, even though some activities can be blocked by the reflective layer. Typically, a highly-solicited agent can continue serving clients, while the suspicious requests or events are blocked until the reflective layer returns a decision. This ‘flexibility’ between the components of the architecture increases the dependability of the agent facing exceptional situations as intrusions.

The architecture allows deployment self-protected agents in the network, where the protection is ensured by the reflective layer. When this protection reaches its limits, the reflective layer then collaborates with CIME.

### 3.2 Architecture of CIME

CIME (Collaborative Intrusion Management Elements) are dedicated intrusion handling elements in the network. They concentrate relevant information from the self-protected agents, agent platform and other sensors, process it to detect the intrusions in the network and assist the reflective layers of protected agents in their self-protection tasks. CIME consist of two functionalities for direct support of protected agents and for

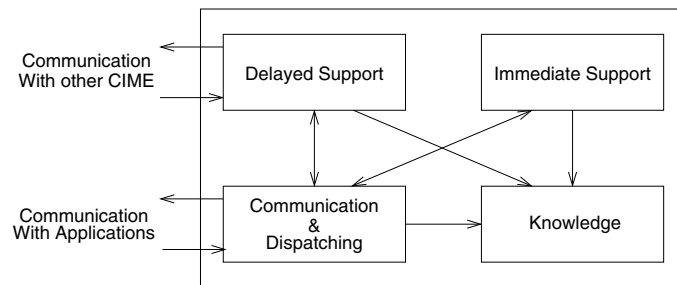


Fig. 3. Architecture of CIME in the network

collaboration with other CIME in the network. CIME receive requests and feedbacks from agents that initiate the first functionality. In reply to requests, CIME can provide two types of support.

- **Immediate support.** CIME provide agents with an immediate solution to their requests. This type of support targets particularly known intrusions. For example, CIME can confirm or dis-confirm that an entry is a phishing attempt.
- **Delayed support.** CIME collaborate with other CIME and, in last resort, with the system administrator to provide agents with solutions when available and possible. This type of support targets unknown intrusions, i.e. new suspicious signature, unusual traffic, etc.

A directory service allows CIME and agents to discover each other, and it also allows CIME to coordinate, share knowledge about intrusion history and possible remedy, and engage in their collaboration protocols. The immediate support is built on the current knowledge of CIME. When this knowledge is sufficient to reply to the agent, the CIME does not need collaborate and can reply immediately. The delayed support is triggered whenever the contacted CIME lacks knowledge for an immediate reply and requires additional knowledge from other CIME or, in last resort, the system administrator. The support actions from CIME can finally be of two distinct kinds, namely *consumable* and *persistent*. Consumable actions are direct fix such as a yes/no reply to a question about phishing attempts. Consumable actions are dynamic in the sense that they are contextual fix: A phishing attempt against a web-site is not permanent, and the web-site URL might change from a ‘reputable’ address to potential danger. Conversely, persistent support actions are typically permanent patch that are deployed in agents, e.g. to fix a buffer overflow. Their deployment prevents the escalation of a contamination by runtime permanent introduction of remedy.

## 4 Collaboration Schemes

CIME agents support the self-protected agents in case of intrusion attempts, and their strength is in the collaboration with other CIME executing on the network to share knowledge and services related to intrusion history or new remedy manually injected in the network by system administrators. The present section aims to describe the main interaction protocols involved. The different protocols show how the different actors of the network interact in realizing the high-level process described in Fig. 1. The notation follows the UML 2.0 specification [6].

We assume that self-protected agents register on startup with a CIME agents on the network, and that all CIME agents register with a directory service, also on startup. These registration protocols are also not represented in this paper, as they can rely on standard directory services of agent platform. The registration of agents with CIME agents is static in the following, but a dynamic version could be considered as well. CIME agents provide the directory service with information to publish, so that they are visible to other CIME agents when collaborations are required. The published information is a list of pairs (*id, symptom*), where *id* is the identifier of the registered CIME and *symptom* is a specific type of intrusion the CIME is capable to deal with.

### 4.1 Protocol Between Protected Agent and CIME

Fig. 4 shows the interaction protocol between the agent that has detected a suspicious input and the corresponding CIME agent. When an agent receives an input, it

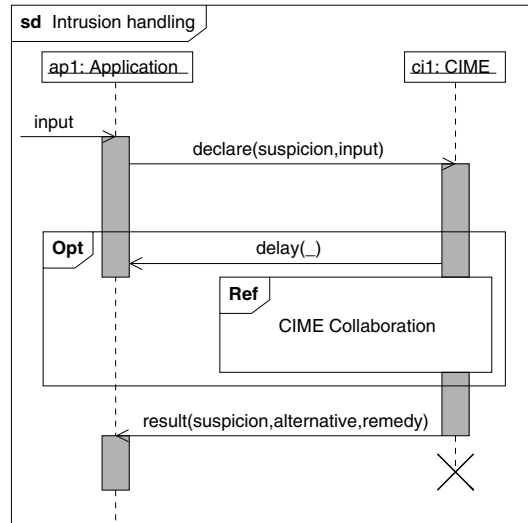


Fig. 4. Interaction protocol between protected agent and CIME

first checks for suspicious clues among the type of intrusions supported in this framework. The finding of such suspicious clue triggers the protocol in Fig. 4. The agent declares the suspicion to the CIME with the appropriate message. If the CIME can provide immediate support, it does not execute the optional sequence (Opt) and directly provides the protected agent with a resulting remedy. For the declared suspicion, the field `remedy` is `nil` if the declared suspicion is a false positive. Otherwise, `remedy` is not `nil` and points to a handling procedure (e.g. deploy a patch, yes/no for phishing). The `alternative` parameter serves to inform the agent that an intrusion attack occurred, but that the type of attack differs from the original suspicion, as piggy-back acknowledgment. The `remedy` parameter then also points to the appropriate handling procedure. If the CIME is unable to provide immediate support, it will first enter the optional sequence to inform the agent of a delay in the support, and initiate the CIME Collaboration protocol. This latter protocol describes how CIME collaborate and is described in the next section. Upon the completion of collaboration, the CIME sends a result to the agent. In case the collaboration did not produce any outcome, the `remedy` informs the agent to apply a default handling procedure, i.e. to block the activities related to the suspicious input. The case is then escalated to human operator.

Algorithm 1 sketches the evaluation process of the CIME agent which decides between immediate and delayed support. This algorithm is executed on reception of a `declare` message, and its output determines whether the optional sequence will be executed. The algorithm assumes that the knowledge base KB of the CIME is consistent (i.e. one handler and only one for a type of intrusion), and each entry of KB contains a type of intrusion and the corresponding handler. The algorithm takes the suspicion and input parameters from the agent message as input, and it provides a remedy, alternative and message as its output. After initialization of the output parameters, the CIME

```

Input: suspicion, input
Output: remedy, alternative, message
1 remedy ← nil;
2 alternative ← nil;
3 message ← nil;
4 foreach entry ∈ KB do
5   | if applicable(input, entry) then
6   |   | remedy ← entry.handler;
7   |   | if match(suspicion, entry) then
8   |   |   | alternative ← entry.type;
9   |   |   end
10  |   | Break for loop;
11  | end
12 end
13 if remedy is nil then
14 |   message ← 'delay';
15 else
16 |   message ← 'result';
17 end

```

**Algorithm 1.** Evaluation Algorithm of CIME

browses its KB in search for appropriate handling procedure. The search tests whether the current entry is applicable to the input. If the test is positive, the reference to the handler of the entry is stored in *remedy*. However, the applicable entry can correspond to another type of suspicion. If such situation occurs, the type of the entry is stored in *alternative* to inform the agent and entail a revision of its evaluation process. When an entry is applicable, the *for* loop is ended. A final test verifies whether the *remedy* has changed. If unchanged (*nil*), the *message* becomes 'delay', i.e. the algorithm ends with a delayed support and the CIME will execute the optional collaborative sequence in the protocol. If *remedy* has changed, the *message* becomes 'result' and the algorithm ends with an immediate support. The CIME will then not execute the optional sequence in the protocol.

Besides addressing the immediate requests from the agents, two more trivial interaction protocol exists between the self-protected agents and CIME elements. The **alarm** protocol is used when the reflective layer of an agent encounters a known threat, resolves the situation locally and informs the CIME about the attack occurrence, so that the suspicious traffic can be identified by the CIME. In case of the above protocol, the suspicion declaration message received from the agent constitutes an implicit alarm.

The **patch** protocol allows the CIME elements to act proactively, and to dispatch the corrections of agent's vulnerabilities before any individual agent is attacked. Such behavior is typically triggered by the attack on a single agent from a multitude of similar agents running in the network, and the CIME may decide to protect all vulnerable agents before they get attacked. This protocol can also be triggered by CIME collaboration or a manual intervention from system administrator. This protocol is simply a simplified stand-alone action from Fig 4, when the result (with a permanent solution) is sent

without any previous suspicion declaration message. This protocol, as well as the alarm, can both be implemented using simple broadcasts, or more complex subscribe-inform type protocols, based on the directory information.

### 4.2 Protocol Among CIME

The protocol in Fig. 5 describes the collaboration patterns between CIME. The collaboration starts when a CIME agent needs the information from others. The first fragment (Opt) is optional and serves to discover other CIME in the network when none are

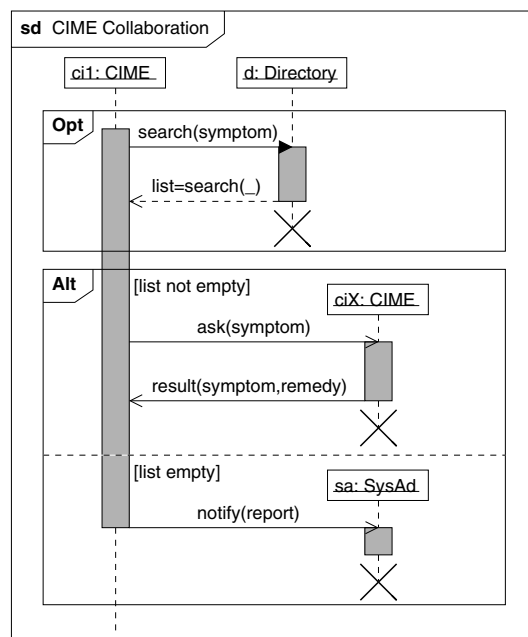


Fig. 5. Collaboration protocol among CIME

known. The CIME contacts the directory service available in the network with a *symptom* parameter. The parameter serves to compile a *list* of CIME that can deal with the particular known symptom<sup>1</sup> (which is related to the type of suspicion and input received by the agent). When the CIME already knows the identity of the peer to contact, it starts with a non-empty *list* parameter. If the *list* of acquaintance is not empty, the CIME will ask to a member of the list for a remedy to the symptom. Different heuristics exist for choosing the member in the list, and we were not concerned with the choice of a particular one. The peer CIME eventually replies with an appropriate remedy. In such case, the requesting CIME revises its knowledge to refer directly to this peer for the same symptom (local storage is also possible, but we would like to avoid overloading

<sup>1</sup> Using the standardized identification, for example from: <http://cve.mitre.org/>

network nodes with redundant information). The CIME can then return to the interaction protocol with the protected agent to provide the freshly acquired remedy. If the *list* is empty, the CIME cannot automatically process the intrusion case, and it is safer to refer to the system administrator by notifying a report based on the identity of the protected agent, the symptom, and a time-stamp for tracking. In addition, the CIME will consistently terminate the protocol with the agent by asking to apply a default handling procedure. The agent should interrupt the activity related to the flawed input, waiting for human intervention. The agent can then proceed with other activities to avoid blocking a network element.

## 5 Related Work

The proposed system can be classified as a host-based intrusion detection system. Its protected agents are based on the hosts (within agent platforms), and the CIME agents are based mostly on the observation of agent/host behavior. On the other hand, the fact that the messages can be inspected and filtered by network-based or low-level elements represent a network-based capability, making the conceptual architecture essentially hybrid, as are many current IDS systems [7,8,9]. Besides the research systems, many commercial solutions [10] aim to filter the traffic on the host level (i.e. personal firewalls), to analyze suspicious behavior of application and users (e.g. Tripwire), or to combine these tasks with other inspection methods. When compared to these approaches, our system differs by working on higher level due to its integration with specific application agents – this allows us to detect the attacks that respect the protocol and low-level system integrity, but attack the system knowledge or reasoning instead. The integration via exception handling and separate reflective layer makes a clear distinction and spares the baseline system from the additional complexity related to self-protection tasks.

The exception model is an high-level version of exception handling mechanisms in programming languages, with an appropriate semantics. The semantics differs from the ones developed in programming languages (e.g. Java-like or Eiffel-like) and it is more related to the work done in component-based software development and architecture description languages. Dellarocas proposed to introduce ‘sentinel components’ in the development process of a component-based system to cope with exceptions that can occur in the composition of COTS [11]. Similarly to our approach, this work deals with exceptions that result from inter-component events (e.g. RPC, message). The major difference is the set of collaborative capabilities that introduces more functionalities in our model of exception, and potentially expands the panel of intrusions that can be handled. Architecture-based exception handling introduces another semantics of exception, where the architecture of a system evolves at runtime to compensate exceptional situations [12]. This semantics of exception is also related to the one presented here, but it also does not include the collaborative functionalities. Intrusion detection techniques are being standardized by the IETF [13]. The related working group has defined a message format for information about intrusions [14] and specific protocols between peers on a network to cope with intrusion detections [15]. The interaction protocols proposed in this paper are application-oriented and they were designed so that they can be deployed on top of standard lower-level protocols, notably the ones developed by the IETF.

## 6 Conclusion

Intrusion detection and handling is a significant security issue for distributed multi-agent systems. As these systems often fulfill mission-critical functions [16], the resistance to intentional attack shall be an integral part of the design. Intrusions must be identified, contained and thwarted in real-time, even though they may use complex strategies or distributed threats. The system we propose offers a close integration with application logic it protects, therefore being able to detect the attacks by means of exception handling, but in the same time uses proven exception handling techniques to make a clear separation between the application code and meta-algorithm in the protection layer. The distributed CIME agents can gather and correlate the information from several sources and increase the quality of system output by partially eliminating the false positive/negative results. In such cases, the additional information about the overall system status is used to complement the suspicion declaration received from the protected agent. CIME agents can provide immediate and delayed support to protected agents, and they are able to supply either direct feedbacks (yes/no answers) or long-term remedies (permanent patches). Delegation of exception handling to the dedicated CIME agents is an essential characteristics of the system (it allows CIME to reason globally) with sufficient data and with the knowledge of the associated network traffic. This more complete vision allows CIME to react more efficiently when encountering threats with no known solution: A CIME can use its knowledge base to generate a filtering rule that can remove the malicious messages before they reach vulnerable agents.

In our future work, we will extend the presented approach in two directions – we can generalize the approach to gather wider range of alarms from protected hosts, and to correlate these alarms with the network traffic observations in order to automatically prevent the spread of malicious code in computer networks [9]. On the other side of the application spectrum, we intend to deploy the solution to protect a resource-constrained sensor networks against misuse.

## Acknowledgment

Martin Rehak and Michal Pechoucek gratefully acknowledge the support of the current project by Army Research Laboratory projects N62558-05-C-0028, N62558-07-C-0001 and N62558-07-C-0007. Martin Rehak would also like to acknowledge the support of the National Institute of Informatics in Tokyo during his inspiring visit.

## References

1. Šišlák, D., Pěchouček, M., Reháček, M., Tožička, J., Benda, P.: Solving inaccessibility in multi-agent systems by mobile middle-agents. *Multiagent and Grid Systems* 1, 73–87 (2005)
2. Platon, E.: Model of Exception Management in Multi-Agent Systems. PhD thesis, National Institute of Informatics, Sokendai, Tokyo, Japan (2007)
3. Goodenough, J.B.: Exception Handling: Issues and a Proposed Notation. *Commun. ACM* 18, 683–696 (1975)
4. Parnas, D.L., Würges, H.: Response to undesired events in software systems. In: *International Conference on Software Engineering*, pp. 437–446 (1976)

5. Reháček, M., Tožička, J., Pěchouček, M., Železný, F., Rollo, M.: An abstract architecture for computational reflection in multi-agent systems. In: *Intelligent Agent Technology, 2005 IEEE/WIC/ACM International Conference*. Number PR2416 IEEE, Los Alamitos (2005)
6. : Unified Modeling Language Specification, UML version 2.0 (August 2005) (Accessed in December 2006), <http://www.omg.org/docs/formal/05-07-04.pdf>
7. Axelsson, S.: *Intrusion detection systems: A survey and taxonomy*. Technical Report 99-15, Chalmers Univ (2000)
8. Keromytis, A.D., Parekh, J., Gross, P.N., Kaiser, G., Misra, V., Nieh, J., Rubenstein, D., Stolfo, S.J.: A Holistic Approach to Service Survivability. In: *Workshop on Survivable and Self-Regenerative Systems (SSRS)*, pp. 11–22 (2003)
9. Reháček, M., Pěchouček, M., Prokopová, M., Foltýn, L., Tožička, J.: Autonomous protection mechanism for joint networks in coalition operations. In: *Knowledge Systems for Coalition Operations 2007, Proceedings of KIMAS'07* (2007)
10. Northcutt, S., Novak, J.: *Network Intrusion Detection: An Analyst's Handbook*. New Riders Publishing, Thousand Oaks, CA (2002)
11. Dellarcas, C.: Toward Exception Handling Infrastructures in Component-based Software. In: *Proceedings of the International Workshop on Component-based Software Engineering* (1998)
12. Issarny, V.; Banâtre, J.P.: Architecture-based Exception Handling. In: *Hawaii International Conference on System Sciences* (2001)
13. : IETF Intrusion Detection Exchange Format Working Group (2007) (Accessed in January 2007), <http://www.ietf.org/ids.by.wg/idwg.html>
14. : IETF Intrusion Detection Message Exchange Format (2007) (Accessed in January 2007), <http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-16.txt>
15. : IETF Intrusion Detection Exchange Protocol (2007) (Accessed in January 2007), <http://www.ietf.org/internet-drafts/draft-ietf-idwg-beep-idxp-07.txt>
16. Maturana, F.P., Tichý, P., Staron, R.J., Slechta, P.: Using dynamically created decision-making organizations (holarchies) to plan, commit, and execute control tasks in a chilled water system. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) *DEXA 2002. LNCS*, vol. 2453, pp. 613–622. Springer, Heidelberg (2002)