Security Event Monitoring in a Distributed Systems Environment

Lukasz Kufel | Poznan University of Technology

Availability and event monitoring of IT systems is necessary for a secure operational environment. This article presents the design and implementation of two security event monitoring approaches in a distributed systems environment.

Computing systems' continual growth and diversification require IT professionals to use a comprehensive management system to maintain their infrastructure. Such systems need a monitoring module that oversees the IT environment and provides proactive notifications from a self-contained monitoring application to assist, for example, in identifying unauthorized access to a company's resources.

Many commercial and noncommercial security information and event management (SIEM) tools on the market offer availability and event monitoring of IT systems.[1–3] These applications are based on two traditional monitoring approaches—agent based and agentless. In this article, I examine these two approaches and present a novel customizable monitoring approach—order-based monitoring (OBM)—that combines them.[4]

Motivation

In February 2009, a company asked me to identify a monitoring approach that allowed customizable implementation and provided security event statuses for 130 servers in less than 10 minutes.[5] (The company proposed the 10-minute frequency after reviewing the US National Institute of Standards and Technology's *Guide to Computer Security Log Management*.[6]) The company also wanted the monitoring solution to provide server availability information. Thus, the selected solution had to

- monitor the system's security events,
- monitor the system's operational events,
- monitor distributed systems' availability, and
- allow customizable and lightweight implementation.

To fulfill these requirements, I examined existing fault-monitoring approaches.

Traditional Business Approaches

Again, the two most popular monitoring approaches currently available on the market are agent based and agentless. *Agent* is defined as "an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives."[7]

An agent-based approach requires installation of additional software on each monitored system.[8,9] Critical business systems use this approach to examine in-depth or unique metric information for incident root-cause analysis. However, the agent-based approach can be difficult to manage and maintain: it often requires server-side resources, and distributed systems' availability monitoring won't work without additional modules. Ciprian M. Dobre and colleagues described a lightweight and platform-independent implementation of this approach that focused on monitoring a predefined set of metrics.[10] As Caroline C. Hayes pointed out, "All types of agent face a similar set of challenges. Many of the issues [in an agent-based system] centre on how to represent, describe,

 Copublished by the IEEE Computer and Reliability Societies

and control systems of agents, and how to get them to co-operate effectively."[11]

An agentless approach relies on built-in system protocols such as the Simple Network-Management Protocol (SNMP) or technologies such as Windows Management Instrumentation (WMI) that offer a standardized way to access information from various systems in an enterprise environment.[12] Therefore, implementing an agentless solution doesn't require additional software, making it easier to implement and maintain in a distributed systems infrastructure. Because there are no agents, updating the monitoring platform considers only the monitoring system. And because of the architecture design, availability monitoring doesn't require additional modules. However, the agentless approach is limited in the granularity and scope of data it can monitor. Identifying an incident's root cause requires additional diagnostic tools.

Neither approach provides a comprehensive solution. The agent-based approach offers an in-depth view, but only for a particular monitored system.[13] In addition, configuration and maintenance aren't easy tasks. The agentless approach is a lightweight solution that gives a holistic view of the IT infrastructure, but it offers only a general overview of the monitored environment.

Today's SIEM market offers solutions with an agent-based approach, an agentless approach, or both approaches working together. The tools target two markets: small- to medium-sized (less than 500 devices in the network) and medium- to large-sized (more than 500 devices). I reviewed all 25 tools evaluated by Gartner to verify following criteria:[3]

- *Availability monitoring*. This feature helps identify resources' accessibility status.
- *Lightweight installation*. I checked each tool's implementation for complexity and resource consumption, classifying the product as lightweight when installation files were less than 200 Mbytes and not appliance based.
- *Freeware software and/or shell commands usage*. This feature allows the product to be more customizable and cost-effective.

Table 1 shows the results of my review.

An Order-Based Approach

Which approach fulfills all my requirements? The agent-based approach meets all requirements except availability monitoring and lightweight installation. The agentless approach is easier to put into operation, but querying 130 servers sequentially from one point might cause problems—a collection poll cycle might not have enough time to complete before the next poll begins. This would cause delays and inconsistency in the collected data. For example, there might be 11 data points per hour instead of 12 when monitoring is configured to run every five minutes.

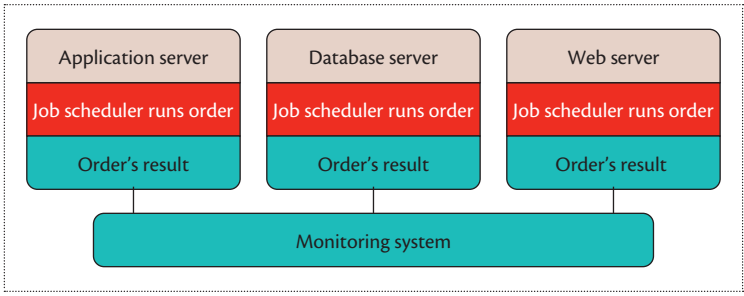

Figure 1. Order-based monitoring architecture. On each of the monitored servers, the server's scheduler invokes monitoring. The results are stored in a predefined location, then collected by the monitoring system.

During my research, I discovered that for every instance of collection, the tools queried the remote system for the same kind of data. Thus, I propose a new order-based approach for distributed systems monitoring. With OBM, the user, or "customer," considers what needs to be monitored and what data must be collected before the monitoring occurs. This point is crucial because it directly influences the amount of data to be maintained. The customer places an order that consists of the system or application metrics to be monitored as well as information about collection frequency (see Figure 1). The order could be implemented as a scheduled task that executes platform shell script or system tools. It's a self-checking system, similar to the observer concept Michel Diaz and colleagues introduced.[14]

OBM integrates the traditional implementations' advantages into one solution. It

- provides in-depth information;
- is lightweight;
- isn't programming language oriented; and
- makes use of built-in resources such as WMI technology, protocols such as SNMP, and system diagnostic commands and online available administrative tools such as Windows Sysinternals.

OBM is similar to a lightweight agent-based approach but differs in code implementation. Agent software is usually written in C, C#, or Java and needs to run in noninteractive mode as a service, whereas my approach doesn't require programming skills or execution in service mode. The monitoring data can be provided using shell command, built-in system tools, or third-party freeware. OBM adds value because it keeps track of the customer's monitoring requirements and gathers no additional data. It uses less of the monitored platform's resources while providing information about the requested metrics.

Table 1. Security information and event management (SIEM) tools comparison.

	Availability monitoring	Lightweight installation	Freeware software and/or shell commands usage
Small- to medium-sized market			
AlienVault SIEM	■		■
CorreLog	■	■	
eIQnetworks SecureVue	■		
IBM TSIEM			
NetIQ Security Manager			
Prism Microsystems EventTracker	■	■	
Quest Software InTrust			
S21sec Bitacora			
SolarWinds LEM (formerly TriGeo)			
Splunk		■	
Tango/04	■		
Tenable SecurityCenter	■	■	
Tier-3 Huntsman			
Tripwire Log Center			
Medium- to large-sized market			
ArcSight			
LogLogic	■		
LogRhythm	■		
netForensics	■		
NitroSecurity NitroView			
Novell Sentinel	■		
Q1 Labs			
RSA (EMC)			
Sensage			■
Symantec SSIM			
Trustwave SIEM			

OBM also differs from traditional approaches in monitoring mode. In an agent-based approach, the system collects data periodically, then pushes the data to the central monitoring console; data transfer happens once information is available. In an agentless approach, data is captured only on request. OBM operates on data prepared prior to collection, which allows it to gather information more quickly than an agentless approach.

I examined 10 characteristics of monitoring approaches:

- *Platform dependency*. When considering implementing a particular monitoring approach, identifying system platforms is important. Platform dependency means special software is necessary for each platform type.
- *Scope of monitoring*. In agent-based and OBM approaches, the scope of monitoring depends on the software and code that those approaches execute. Agentless monitoring is limited to the protocol or technology used—for example, SNMP.
- *Availability monitoring*. This feature allows the monitoring approach to report on a system's availability status, that is, whether a system is online and accessible to other resources.
- *Resource utilization and event monitoring*. This feature indicates the ability to monitor resource utilization such as CPU, memory, and disk space. Event monitoring includes system and security events available on the platform.
- *Monitored data granularity*. Because agent-based and

Table 2. Overview of monitoring approaches.

	Agent-based approach	Agentless approach	Order-based monitoring approach
Platform dependency	Yes	No	Yes
Scope of monitoring	Customizable, on demand	Predefined set of features	Customizable, on demand
Availability monitoring	No	Yes	Yes
Resource utilization and event monitoring	Yes	Yes, limited	Yes
Monitored data granularity	In-depth, full	General, limited	In-depth, full
Monitored data transfer mode	Push	Request and response	Pull
Scripting language proficiency	Not required	Not required	Required
Solution type	Heavy, lightweight	Lightweight	Lightweight
Ready for enterprise	Yes	Yes, for general metrics; no, if custom metrics are required	Yes
Software installation required on monitored system	Yes	No	Yes, for customer requirements; no, for standard system features

OBM approaches are fully customizable, they can provide any type of information from customer-requested systems or platforms.

- *Monitored data transfer mode*. Push mode indicates a time- or event-driven situation, such as when data is sent from the monitored object to the monitoring system. Request and response mode retrieves data from the monitored object only when the monitoring system sends a monitoring request. Pull mode works on the monitoring system and collects already prepared data from the monitored object.
- *Scripting language proficiency*. The OBM approach requires scripting language proficiency because it uses shell commands and freeware system tools.
- *Solution type*. This feature indicates the monitored object's use of resources, such as disk space and memory.
- *Ready for enterprise*. This feature indicates the monitoring approach's readiness for enterprise use.
- *Software installation required on monitored system*. This characteristic identifies additional system administration steps that need to be performed when deploying a monitoring solution.

Table 2 shows the monitoring approaches' characteristics.

Implementation

Here, I examine the OBM and agentless approaches' implementation requirements. I don't include the agent-based approach's implementation owing to its complex deployment and management.[7,11]

Agentless Approach Implementation

A key feature of a traditional agentless approach is that it doesn't require installation of additional software on the monitored system. All common operating systems have built-in monitoring interfaces through which secure access to systems resources is available.

Figure 2 demonstrates an agentless approach's implementation details. Agentless monitoring starts on the monitoring server on which the monitoring system software (MSS) is installed. MSS polls remote servers for one security event and three system events. The first availability checkpoint module ensures the remote server is accessible. If the remote server is inaccessible, the monitoring process finishes. Otherwise, a query is called to send requests to the monitored server. MSS establishes a connection using WMI or Sysinternals PsLogList. Processes operating WMI or PsLogList instructions receive the monitoring request and push it to the system's resources to get the results. Once the data is collected, it's transferred back to the monitoring server. The received response can be directly imported to MSS or written to a local file.

OBM Approach Implementation

The OBM approach concentrates on predefined metrics that need to be monitored. This is a key feature that optimizes monitoring and limits resource utilization. In a typical process, a customer requests an order consisting of specific items. When the order is complete—that is, when requested metrics are checked and their status is available[15]—the monitoring system pulls the order's results, which are available to the customer in the monitoring system and can be delivered by email when necessary.

Figure 3 illustrates a high-level view of the OBM approach's implementation. OBM consists of three layers:

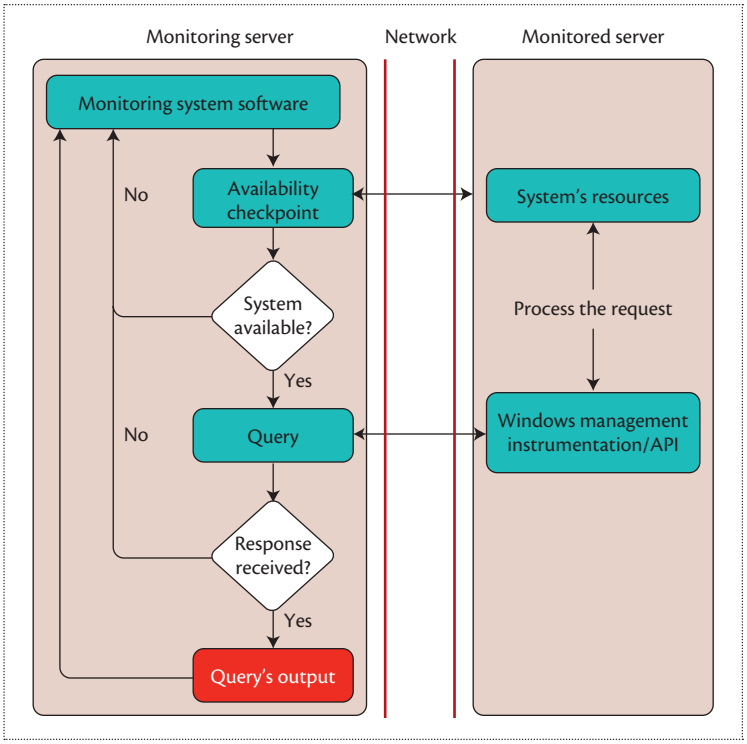

Figure 2. Agentless monitoring implementation details. The monitoring system software validates the monitored server's availability, then requests the status of defined events.

- *Environment.* This layer comprises the IT infrastructure consisting of various types of servers, applications, and services. Each server is equipped with a job scheduler responsible for scheduling server tasks, the order's definition (a script containing customer-requested metrics), and the order's results (a file containing metrics, their values, and a UTC time stamp).
- *Network.* This layer controls the servers' availability monitoring. It also transports the order's collected results to the monitoring system.
- *Collection and presentation.* This layer offers a holistic view of metrics collected during the monitoring process across the entire IT environment. Delivered information is analyzed and stored in a local database, then presented to the customer through a Web interface. Data is maintained according to customer requirements.

Figure 4 shows OBM workflow and implementation. The monitored server and the monitoring server represent the environment layer and the collection and presentation layer, respectively. Both layers exchange data via the network.

I implemented the order's definition in a shell script using system tools to store the order's results in a flat file. The monitored server's job scheduler program executes the script every nine minutes to check the order's definition. As with the agentless approach's implementation, the goal of the order's definition is to explore the server's security and system logs for particular events. This could be done using WMI or Sysinternals PsLogList locally, as opposed to an agentless approach in which it's performed remotely. Once the order's script execution is complete, all the results are stored in the order results file, which resides in a known location, available for collection by another process.

On the monitoring server, another job runs every eight minutes to make the order results file accessible to the MSS. First, the availability checkpoint module ensures the remote servers are running. If a server isn't available, the collection process for this server finishes, and this information is sent to a temporary completion status file. If the server is available, the availability checkpoint module starts the download results module, and the collection process continues.

The download results module copies the order results file from the monitored server's shared folder onto the monitoring server and calculates its age by comparing the time available in the order results file with the monitoring server's time. Then, the time difference information is returned to the order results file (see Figure 5). The file's age determines whether the job scheduler in the monitored server is working as expected (that is, it refreshes the order results file each time it's executed). When the order results files' collection finishes for all monitored servers, the replace results module is called. If a particular order results file couldn't be transferred (for instance, the file was missing or corrupted), the relevant entry is recorded in the temporary completion status log file.

The last module—replace results—overwrites previously downloaded files with new ones and reports that action in the temporary completion status file for further analysis. It's important that file replacement occurs after all the order's results are downloaded and available on the monitoring server. This improves the performance of the entire process and prevents files from being locked.

Finally, the temporary completion status log updates the production file and, together with the order results files, is analyzed and processed by the MSS. There is only one completion status file and as many order results files as there are monitored servers. Because the order results files are replaced each time they're downloaded, updating the MSS isn't necessary. The MSS is configured to read the order results files every five minutes.

Test Results

My work considered two agentless approach implemen-

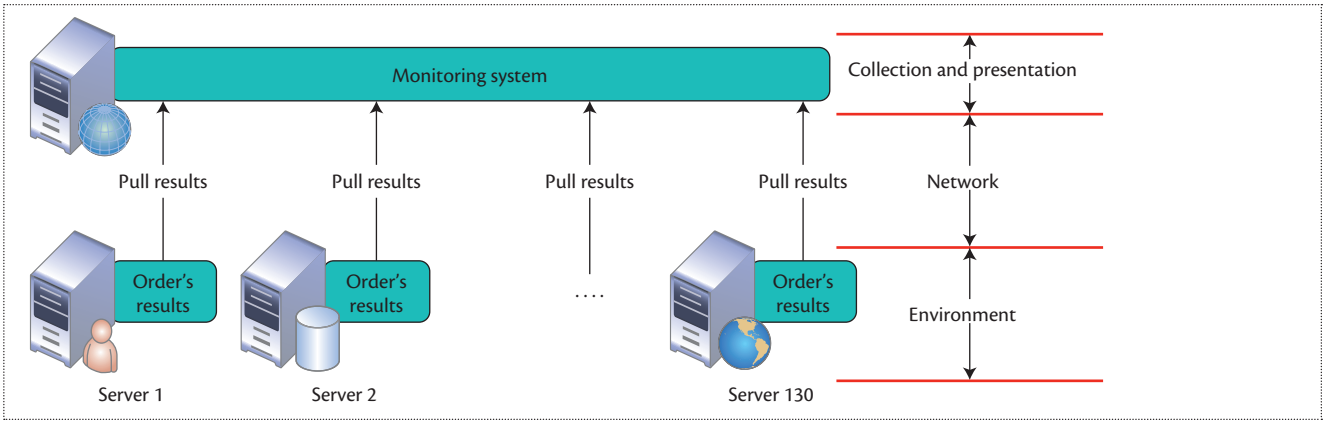

Figure 3. High-level view of an order-based monitoring (OBM) approach implementation. The monitoring system collects the already prepared status of defined events (the order's results).

tations (WMI and PsLogList) and one OBM implementation. I tested 130 servers running Microsoft Windows Server OS using ManageEngine Applications Manager as the MSS. All monitored servers were located in one physical datacenter. However, no solution was limited to OS version, vendor, or geographic location. (Using multiple locations increases the order's collection duration, owing to network latency.)

Using three months of monitoring data, I measured the cumulative time, in seconds, required to carry out the sequential monitoring process, starting with the first server and ending with 130th server (see Figure 6). The cumulative time indicates how fast the security events status can be refreshed for all monitored systems. I assumed the availability checkpoint and query modules executed in the agentless approach whereas availability checkpoint, download results, and replace results modules executed in the OBM approach. The availability checkpoint module used a `ping` command. The query module used WMI technology in the first agentless implementation and the Sysinternals PsLogList application in the second. OBM's download results module used the `xcopy` command to transfer the order results files from the monitored server to the monitoring server, and its replace results module used the `move` command to replace existing files.

OBM delivered events status information much more effectively than the agentless approaches, because it worked on data that was already prepared by the monitored system. OBM's cumulative time is actually the time it takes to copy the order results files from the monitored servers to the monitoring system server. In the agentless approach, the cumulative time also consists of time the monitored system needs to check requested metrics.

OBM achieves my server availability and event monitoring requirements easily. It's most effective for companies with nonstandard monitoring needs that require a customized solution:

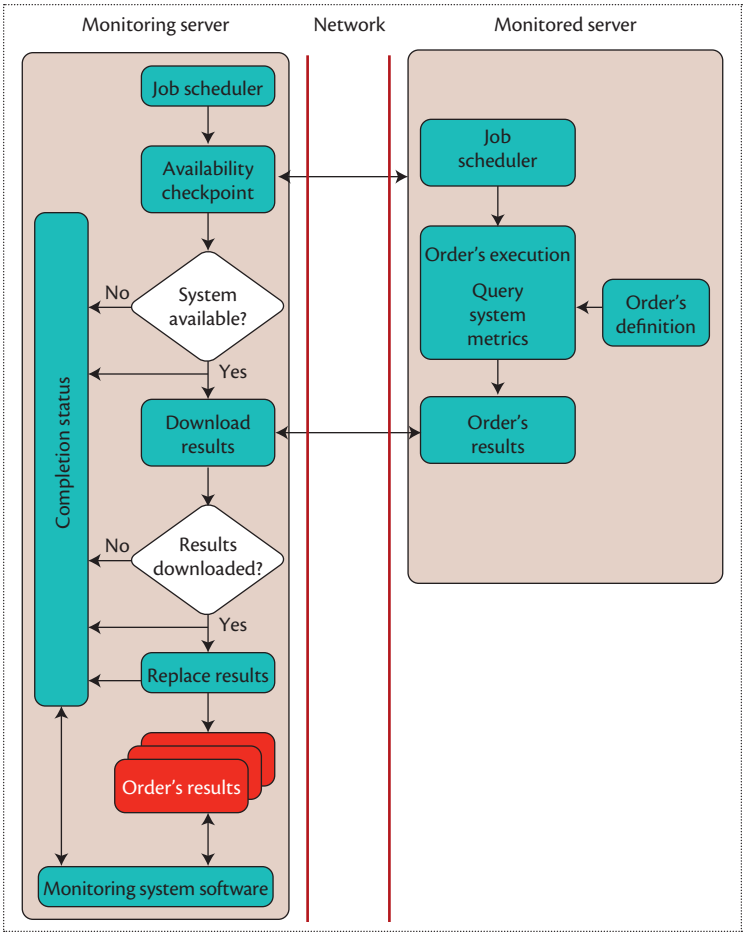

Figure 4. OBM implementation details. The monitored server represents the environment layer and the monitoring server represents the collection and presentation layer. Both layers exchange data via the network. As opposed to the agentless approach, OBM starts on the monitored server.

canwn0049.txt

```
1  <--table Events_Monitoring starts-->
2  Log_Name|Number_of_events|Message
3  Security|0|OK
4  System|0|OK
5  <--table Events_Monitoring ends-->
6  Server_name=canwn0049
7  Script_execution_time=15:49:01 , 16Sep-2009 (Wed)
8  Log_age_in_minutes=1
```

Figure 5. Example of an order results file. The file is structured in the format required by the sample monitoring system software.

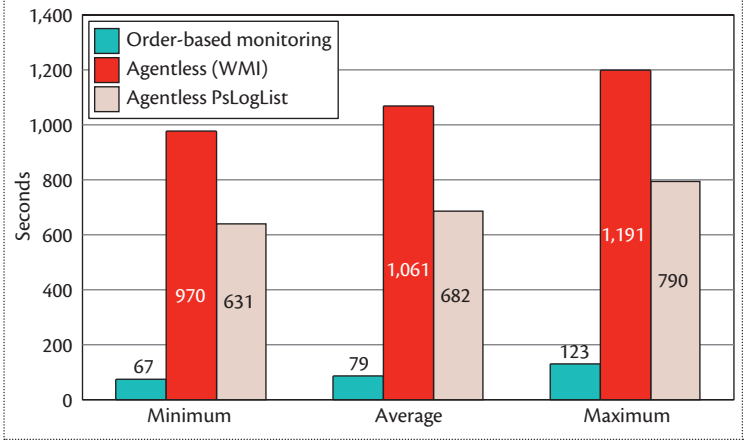

Figure 6. Comparison of monitoring approaches. Cumulative time indicates the base frequency of the event monitoring in a sample distributed systems environment.

- It's scalable to a medium-sized market. Extending the monitoring order's definition by more metrics doesn't increase shipping time to the monitoring system platform.
- It supports little monitoring demand, for instance, if the customer is only interested in a server's CPU utilization and availability.
- It's optimized on the architecture level. For example, when monitoring Unix servers through command-line interface, it uses only one connection to receive the data—results download.
- In my sample implementation, it was able to provide data to MSS at least eight times faster than the traditional agentless approach.

In addition, my approach is more effective and easier to adopt than existing traditional approaches. By combining SNMP protocol possibilities with WMI technology, users can monitor all metrics that Microsoft Windows Server OS offers, as both were designed to provide systems' health status. However, based on my research, it doesn't work effectively on large-scale distributed systems monitored remotely or when customers want to check the metrics frequently. WMI gives an interface to the system and security events, but unfortunately, it fails when agentless monitoring occurs every five minutes for more than 50 systems because a monitoring poll cycle doesn't have enough time to complete before the next poll starts. SNMP is more efficient than WMI but doesn't offer system and security event monitoring and thus doesn't meet my requirements.

Deployment possibilities for OBM include monitoring availability and event log check-out; monitoring system processes and service availability; and monitoring standard servers' capacity metrics, such as file systems, CPU, and memory utilizations. In the future, I plan to implement a demonstrated OBM solution in different geographic sites, taking time zone differences into consideration. ■

Acknowledgments

I'm grateful to the three anonymous reviewers for their valuable comments. Special thanks are due to Jan Weglarz for his insightful remarks and suggestions as well as to Paul Elgar and Roger Egleton for language corrections.

References

1. N. Delgado, A. Quiroz Gates, and S. Roach, "A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools," *IEEE Trans. Software Eng.*, vol. 30, no. 12, 2004, pp. 859–872.
2. B. Ravindran, "Engineering Dynamic Real-Time Distributed Systems: Architecture System Description, Language, and Middleware," *IEEE Trans. Software Eng.*, vol. 28, no. 1, 2002, pp. 30–57.
3. M. Nicolett and K.M. Kavanagh, *Magic Quadrant for Security Information and Event Management*, Gartner, May 2011.
4. W.N. Robinson, "A Requirements Monitoring Framework for Enterprise Systems," *Requirements Eng.*, vol. 11, no. 1, 2006, pp. 17–41.
5. C.-G. Guo, X.-L. Li, and J. Zhu, "A Generic Model for Software Monitoring Techniques and Tools," *Proc. 2nd Int'l Conf. Networks Security Wireless Communications and Trusted Computing* (NSWCTC 10), IEEE CS, 2010, pp. 61–64.
6. K. Kent and M. Souppaya, *Guide to Computer Security Log Management*, US Nat'l Inst. Standards and Technology, Sept. 2006; http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf.
7. N.R. Jennings, "On Agent-Based Software Engineering," *Artificial Intelligence*, vol. 117, no. 2, 2000, pp. 277–296.
8. R. Govindu and R.B. Chinnam, "MASCF: A Generic

Process-Centered Methodological Framework for Analysis and Design of Multi-agent Supply Chain Systems," *Computers and Industrial Eng.*, vol. 53, no. 4, 2007, pp. 584–609.
9. R. Subramanyan, J. Miguel-Alonso, and J. Fortes, *Design and Evaluation of a SNMP-Based Monitoring System for Heterogeneous, Distributed Computing*, tech. report TR-ECE 00-11, School of Electrical and Computer Eng., Purdue Univ., July 2000.
10. C.M. Dobre et al., "An Agent Based Framework to Monitor and Control High Performance Data Transfers," *Int'l Conf. Computer as a Tool* (EUROCON 07), IEEE CS, 2007, pp. 453–458.
11. C.C. Hayes, "Agents in a Nutshell—A Very Brief Introduction," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, 1999, pp. 127–132.
12. P. Bellavista, A. Corradi, and C. Stefanelli, "Java for On-Line Distributed Monitoring of Heterogeneous Systems and Services," *Computer J.*, vol. 45, no. 6, 2002, pp. 595–607.
13. T. Azemoon et al., "Real-Time Data Access Monitoring in Distributed, Multipetabyte Systems," *Int'l Conf. Computing in High Energy and Nuclear Physics*, Journal of Physics: Conference Series 119, 2008.
14. M. Diaz, G. Juanole, and J.-P. Courtiat, "Observer—A Concept for Formal On-Line Validation of Distributed Systems," *IEEE Trans. Software Eng.*, vol. 20, no. 12, 1994, pp. 900–913.
15. M. Zulkernine, R.E. Seviora, "A Compositional Approach to Monitoring Distributed Systems," *Proc. Int'l Conf. Dependable Systems and Networks*, IEEE CS, 2002, pp. 763–772.

Lukasz Kufel is a PhD student at Poznan University of Technology and an IT operations manager at Expedia.com. His research interests include distributed systems monitoring and event processing. Kufel received an MS in computer science from the Poznan University of Technology. He's a student member of IEEE and the IEEE Computer Society. Contact him at lukasz.kufel@computer.org.