STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Artur OPALIŃSKI
Politechnika Gdańska, Wydział Elektrotechniki i Automatyki

ADAPTING A GENERAL TOOL TO MONITORING MULTI-AGENT SYSTEMS THROUGH VIRTUAL HOST LAYER EXTENSTION[1]

Summary. Nagios is a free software for IT infrastructure monitoring. Out-of-the-box it is not suited for monitoring multi-agent systems, because agents may dynamically join and leave the system or change roles. But Nagios' flexible configuration makes extensions possible. This paper presents and verifies Nagios configuration and extensions for monitoring multi-agent systems.

Keywords: multi-agent systems, remote monitoring, virtual host layer

ROZSZERZENIE TYPOWEGO NARZĘDZIA O WARSTWĘ HOSTA WIRTUALNEGO W CELU MONITOROWANIA SYSTEMÓW AGENTOWYCH

Streszczenie. Nagios to bezpłatne oprogramowanie do monitorowania infrastruktury IT. Nie jest ono dostosowane do monitorowania systemów agentowych, gdyż agenty mogą swobodnie podejmować lub zarzucać realizację działań czy zmieniać pełnioną rolę. Bogate możliwości konfigurowania Nagios pozwalają poszerzyć jego możliwości. Artykuł przedstawia i weryfikuje konfigurację oraz rozszerzenie Nagios, umożliwiające monitorowanie systemów agentowych.

Słowa kluczowe: system agentowy, zdalny monitoring, warstwa wirtualnego hosta

1. Introduction

Sets of autonomous entities are considered in research and use of multi-agent systems (MAS). These sets are potentially large and dispersed. They may encompass simple, zero-

[1] This work was co-financed by the European Union within European Regional Development Fund, through grant Operational Program Innovative Economy POIG.02.03.00-00-028/09-05.

intelligence, homogenous agents [1] or complex agents [2,3], possibly playing different roles in problem solving. In general, at any instance of time, a given functionality may be provided by any number of agents, and this functionality is hosted on-demand on occasional computing nodes.

To achieve goals with MAS it may be important to have at disposal an appropriate amount of agents or an appropriate mix of agent services for the task to converge to the solution [4]. Both monitoring and provisioning is therefore a requirement for any distributed multi-agent environment. Unfortunately multi-agent environments do not address these features consistently. Some solutions like UMAP [5] or JADE [7] are based on FIPA standards [16] which requires to run agents in controlled environment. This is interpreted as controlling each agent individual management state changes, not as presenting summary data on agent population(s). Other frameworks, like AgentBuilder [17] provide insight into agencies (agent sets) and allow to manage the multi-agent project as a whole but use proprietary mechanisms which are unsuitable for heterogeneous agent sets. Yet others like NetLogo [18] only offer result visualization. In Swarm [19] the dedicated observer swarms can either draw the results as graphs or direct the resulting data to files.

A more promising approach to heterogeneous agent sets monitoring comes from the realm of computer system and network monitoring. IBM Tivoli line of products [20,21], BMC Event Manager (former BMC PATROL) [22], HP OpenView[23] or open-source tools [24] allow to monitor virtually any service by providing for software extensions called monitoring agents or plugins, and a wealth of communication protocols. Unfortunately the monitoring is aimed at every separate infrastructure item or service, or on the status of a group of them. No simple provisions are available to monitor the count of services or the existence of a specific percentage mix of services.

Nagios [6] (formerly NetSaint) is a free, multiplatform, open-source software, popular for monitoring IT infrastructure. Nagios can monitor virtually any item, independent from the underlying software, by using dedicated plugins. Nagios is not well suited out-of-the box for monitoring MAS, though. Its fundamental assumptions include that every service monitored is inseparable from underlying host and that hosts and their services are static, i.e. services are performed by the same computing node over time. This is in contrast to MAS setups where agents may join and leave the active set dynamically, and may take over various and sometimes multiple roles.

On the other hand, Nagios is very flexible in its configuration, making it a promising target for extending its application, even without need to change its source code.

The goal of this paper is to present a solution to MAS monitoring based on concepts found in Nagios. The purpose of this monitoring is to ensure enough agents in any roles are at

play for the MAS to find a solution. The responsibility of the monitoring system is to inform in an automated way that some assumptions or prerequisites pertaining to the presence or quantity of agents are not met, not to decide what numbers of agents or agent roles are needed to perform the task.

2. Primary Nagios usage

2.1. Simple monitoring scenario

Basic Nagios elements in a simple monitoring scenario are presented in Fig. 1, on the left side, with Service A. Nagios Core caches the configuration information in memory. This basic configuration includes a service (`Service A`) linked with hosts `Host`. A host may also be specified for monitoring without any service associated with it. Service and host probes are called with parameters which may be specified respectively in the service or host definition. The service and the host definitions usually point to different probes. The probes do the actual checks to verify the service or host health in the monitored environment and return information, which is intercepted and reflected in Nagios Core as service or host status, respectively.

The relation between services and hosts is many-to-one. Any service needs to remain in a parent-child relationship with a single host, i.e. services must be defined with exactly one underlying host. The relationship is used to improve the notification schema, i.e. to avoid notifications for services that are brought down from an outage on their host, to ease the failure root cause analysis.

2.2. General configuration remarks

Both host and service checks can be individually disabled by means of the *active_checks_enabled* field – this means that it is possible to only actively monitor the state of the host, or only actively monitor the state of the service, despite their relationship. This feature is used below to simplify the examples – only hosts are actively monitored. The probe commands for services and hosts follow the same API rules. The only difference is that for service checks the return codes allow to distinguish more states. Besides this, there is no inherent difference indeed between monitoring a host or a service in regard to the solution presented.

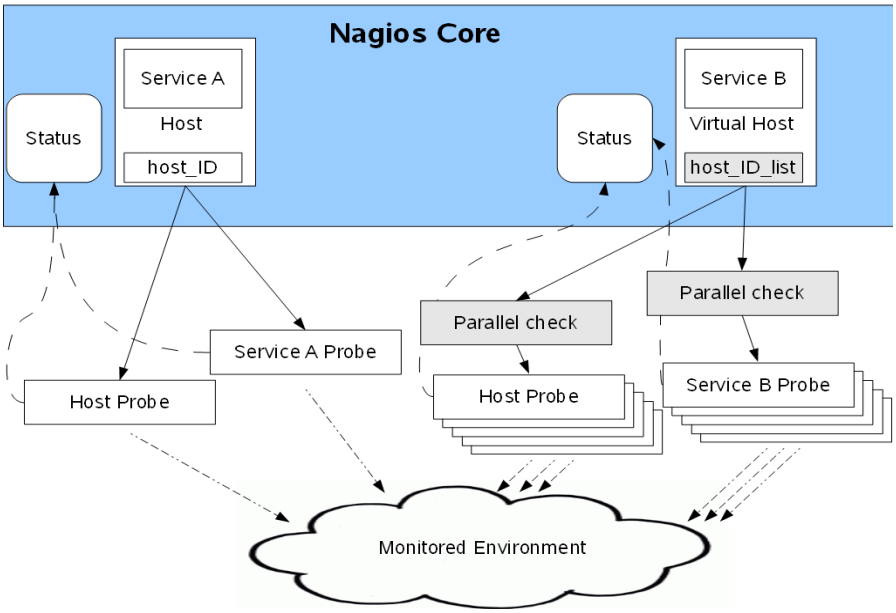

Fig. 1. Monitoring scenarios with Nagios. Service A on Host is monitored in a typical way. Service B on VirtualHost is monitored using virtual host layer

Rys. 1. Monitorowanie z użyciem Nagios. Usługa A na hoście Host jest monitorowana w podstawowy sposób. Usługa B na hoście VirtualHost jest monitorowana za pośrednictwem hosta wirtualnego

2.3. Shortcomings for MAS monitoring

Parent-child relationship is right for monitoring essential services in a relatively static environment, where each service instance is known in advance and where the service is performed by a specific host. Nagios then issues the service and host checks at scheduled times, and the appropriate check commands are run with the corresponding host address as parameter, so a specific host, or a service running on this host, is checked.

This schema does not fit well in monitoring MAS, where system configuration and functionality is dynamic. In MAS, in course of solving a problem, agent code may migrate between various nodes. Agents may also assume various roles and thus provide various functionality.

In Nagios vocabulary, functionality provided by agent is a service. To employ Nagios to correctly monitor such dynamic services, the parent-child relationship should be disabled. Nagios should allow a service to disappear from its current node and perhaps appear on any other node or nodes. Monitoring process should recognize such migrations and the resulting changes in service instant numbers as natural fluctuations in MAS state, as long as the number of each service instances at any time is greater than a threshold set for this service.

3. Solution

Monitored services and nodes, check commands and other configuration objects are separately defined in ASCII text files. Nagios configuration files use proprietary, publicly documented syntax [8]. Despite that, modifying Nagios configuration files on-demand to track the changes in MAS is out of question, as Nagios only reads these files once, during start-up, it immediately verifies them for correctness, and then it caches the data in its internal database. This database has no general public interface, but its contents can be influenced indirectly to a limited extent.

The following architectural elements of Nagios are applicable to extending its capabilities to MAS monitoring: plugins, object definitions extensions with *custom object variables*, external commands, performance data collection. Their role in the solution is described later

3.1. Static virtualization of host layer information

To remove the limitations resulting from the static addressing and static parent-child relationship between host and service, virtual host concept is introduced.

There is and *address* field in every host definition. The value of this field gets expanded as the host address macro $HOSTADDRESS$ and passed as a parameter to the probe command. The host *address* field is defined loosely in the documentation. While it it mentioned that the primary use of this field is to store the IP address, it also stated that no assumptions are made by Nagios on the content of this field – it is just passed to the probe command and it is left to the probe to interpret this value [8]. Documentation goes as far as to state that the *address* field is generic – it can contain anything from an IP address to human-readable driving directions – whatever is appropriate for the user's setup [9].

By pursuing this openness, instead of a single address, the *address* field may be used to store a list of host IP addresses, or a list of host identifiers in general. This allows to change a real host definition into a virtual host definition. It will allow to monitor a service without tying its definition to a specific IP address, i.e. it will allow to monitor a functionality in a system consisting of many hosts, which are seen as a whole. There is nothing to prevent many services to be based on the same virtual host and therefore to monitor different functionality aspects in the whole system. The change in monitoring with virtual host layer is depicted in Fig. 1 for Service B.

The host identifier list is called host_ID_list. A new command needs to be provided and defined as the check command. This command, called *parallel_check,* interprets the list of host identifiers passed to it as parameter, splits it into individual host identifiers and calls multiple instances of the probe commands, in parallel, each with a single host identifier.

Therefore no modifications are required for the original probe commands. The *parallel_check* is required to obey the Nagios requirements for plugins, and should accept the results of probe commands according to the plugin API [10, 11]. The *parallel_check* should accept at least the following information, which needs to be passed as call parameters according to Nagios standards:

- The value of the macro $HOSTADDRESS$, i.e. the list of host identifiers.
- The full form of the original probe command, including its execution path and parameters. A placeholder string (called HOSTADDRESS, in the examples below) marks the parameter to be filled in by parallel_check with the correct host id value for each probe when calling it.
- The threshold defining how many positive probe responses are regarded as overall positive result for the virtual object checked. This number can be a percentage or an absolute number, or any other value interpreted by parallel_check. The threshold in the examples below is expressed in the form of absolute number of positive results expected, with the value zero meaning that all responses are expected to be positive.

Except for a generic *parallel_check* command as described above, the following configuration introduces the name of the host (VirtualHost), and the *address* field containing a list of host identifiers:

```
define host{
       host_name              VirtualHost
       alias                  host for service A
       address                192.168.10.2,192.168.10.3 ; extended to list
       check_command          parallel_check!0!check_ping -H HOSTADDRESS
       . . .                  ; rest of configuration omitted
    }

define command{
       command_name  parallel_check         ; definition of the new check com-
mand
       command_line  /usr/local/bin/parallel_check $HOSTADDRESS$ $ARG1$ $ARG2$
    }
```

3.2. Dynamic updates to virtual host layer

The former static method of host layer virtualization requires that host identifiers of all potential hosts are listed up front, what results in all hosts being actively checked for the expected functionality. While entirely sufficient in many cases, this may be inconvenient, if it is expected that only a fraction of the total number of hosts listed holds active agent code during some periods of work. Checks may be spared and their resource consumption may be lowered, if it could be known in advance that a functionality can not be available on some hosts because the prerequisites are not met, for example a host is not reachable, or no agent registered on this host.

To tune the solution further, the virtual host layer information should be updated dynamically, to only include host identifiers of the hosts which are likely to provide a given functionality. This allows to avoid unnecessary checks for functionality when prerequisites are not met, but it shifts the problem to checking prerequisites instead. This requires to define another object (Host B in Fig. 2) witch its own check method to check for prerequisites for VirtualHost A. So dynamic virtual host layer updates are only meaningful if checking for prerequisites is lighter than checking for the functionality aspects of interest.

Nagios accepts external commands which may influence its state [12, 13]. In Nagios database, the *address* field of the host definition is static, as is most part of the configuration. Therefore dynamic updates must be based on other storage. Fortunately Nagios allows *custom object variables* in object definitions and its external command set allows for dynamic updates by providing a command to change the values of the *custom object variables:*

```
CHANGE_CUSTOM_HOST_VAR;<servername>;<custom_object_variable_identifier>;<new_cus
tom_object_variable_value>
```

To provide the external command with the complete host identifiers list, it is sufficient that the probes called by *parallel_check* of Host B output the host identifiers when check results are positive. Then *parallel_check* can intercept this output, build a complete, current list of valid host identifiers, and return it to Nagios Core. In Nagios Core a performance data collection command can be defined, which parses check commands output, by the following lines in the main configuration file [14]:

```
host_perfdata_command=/usr/local/bin/PerfData  ; perf. data collection command
process_performance_data=1              ; enable performance data processing
```

The performance data collection command may then issue the CHANGE_CUSTOM_HOST_VAR external command with the current list of valid host identifiers. Nagios can be instructed to processes the external commands as often as possible by setting in its main configuration file [14]:

```
check_external_commands=1         ; enable external commands
command_check_interval=-1         ; check as often as possible
```

This ensures that list of host identifiers gets updated after every check and performance data collection command run. To allow for quicker update of host identifier list, the external command could also be issued from *parallel_check* (not depicted in Fig. 2).

The object configuration for dynamic updates to virtual host layer should be as follows:

```
define host{
       host_name        VirtualHostA
       address          anything ; field unused
       _ID_LIST         anything ; custom obj. var., populated automatically
       check_command          parallel_check!0!/bin/check_print HOSTADDRESS
       active_checks_enabled  1         ; checks are enabled
       process_perf_data      0         ; performance data processing disabled
       . . .                            ; rest of configuration omitted
    }
```

```
define host{
       host_name        VirtualHostB
       address          anything ; field unused
       _ID_LIST         192.168.10.2,192.168.10.3,192.168.10.5 ; all hosts re-
garded
       check_command          parallel_check!0!/bin/check_ping -H HOSTADDRESS

       active_checks_enabled  1         ; checks are enabled
       process_perf_data      1         ; performance data processing enabled
       . . .                            ; rest of configuration omitted
   }
```

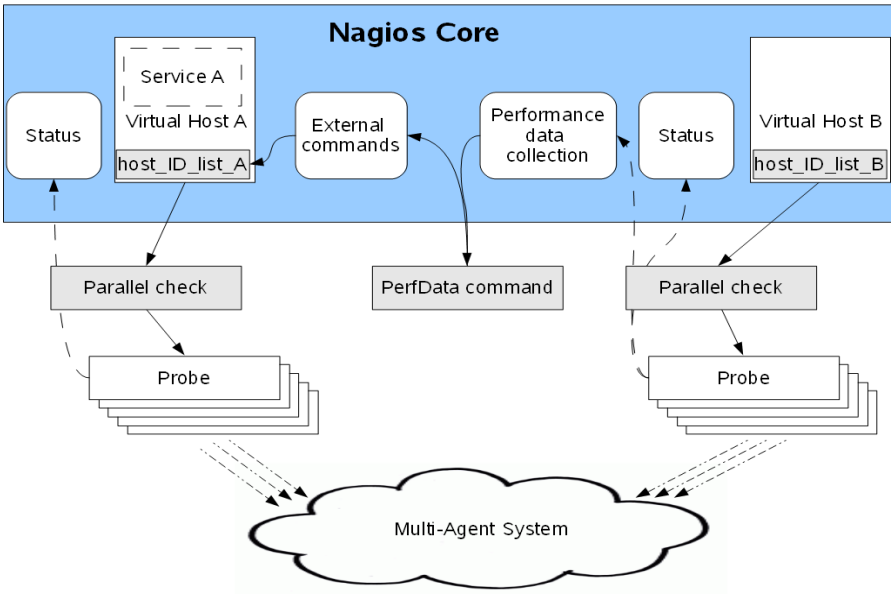

Fig. 2. Monitoring Service A on dynamic VirtualHost .Virtual host layer information updated based on prerequisites checked through static VirtualHost B

Rys. 2. Monitorowanie funkcji Service A na dynamicznym hoście VirtualHost. Informacja w wartswie hosta wirtualnego jest aktualizowana na podstawie kontroli prerekwizytów przez statyczny wirtualny host VirtualHost B

It is of course entirely possible to also construct VirtualHostB on the base of another VirtualHostC and another prerequisites, to create a chain of virtual hosts. Each virtual host in a chain encompasses a subset of host identifiers of its preceding virtual host.

4. Verification

3 PCs with Linux were used as computational nodes for agents and one more computer as the central Nagios monitoring console. The agents checked were SNMP agents. If the host was reachable, the agent was considered providing its services if SNMP returned the information requested (system uptime was examined). The configuration was like in Fig.2. Static VirtualHostB has been defined as consisting of 3 PCs, using *check_ping* as the probe command to check reachability, with the threshold set to 2. The host unreachability was simulated by plugging out their network interface cable. The dynamic VirtualHostA was constructed on

the prerequisite that a host is reachable; a *check_print* command was used to monitor it, which always returns success, and outputs to a file current time and the host identifier obtained as parameter. Verification results are presented in Table 1. Number of host identifiers used by VirtualHostA probes reflects the number and identifiers of hosts reachable, what demonstrates that the dynamic updates to the virtual host layer work. Also the status of the static VirtualHostB correctly reflects the number of nodes meeting the prerequisite.

Table 1

Verification results

No	Number of PCs reachable	VirtualHostB status	Number of distinct host identifiers used by VirtualHostA probes
1	3	OK	3
2	2	OK	2
3	1	Not OK	1
4	0	Not OK	0

The Nagios configuration scripts, the *parallel_check* command, and the *check_print* commands used during verification are available under [15].

It was transparent during monitoring of the agent functionality, which agents responded – only the total number of responses counted. Therefore any agent migrations during testing, even coss-platform migrations, would yield the same results in stable conditions. No migrations were tested due to homogenity of the verification environment.

5. Conclusions

A MAS monitoring extension and configuration to freeware Nagios has been worked out. Agents' joining and leaving the system, or changing roles between adjacent checks are accepted as legitimate fluctuations in MAS state, as long as the number of each service instances at any time is greater than a threshold set for this service. Monitoring comprises of discrete checks scheduled for every functionality, with a freely configurable frequency.

Solution is based on Nagios public interfaces, what ensures maintainability and does not disable any of its existing functionality. Probe commands are the only part of the monitoring solution which may need to be tailored to a given multi-agent setup, to check the agent functionality actually required. This is in line with Nagios architecture where the probe commands are considered external plugins.

It is wort to stress, that while the verification has been conducted using IP addresses as host identifiers, the solution will work as well for other identifiers: agent code may be hosted

in local processes or in Java virtual machines, etc. The initial list of host identifiers can be either predefined statically, or can be imported from some promising sources, like the logs of DHCP server or the cache of the network switch serving the agent network.

It would be advantageous to address also agent provisioning. Provisioning is the process of configuring the infrastructure to provide (new) services, which may be very useful when many agents are at play and numerous scenarios are to be tested. Nagios does not address provisioning in its out-of-the box configuration – but its event handlers mechanisms may be further investigated to achieve this.

BIBLIOGRAPHY

1. Miller P.: The Smart Swarm: How understanding flocks, schools, and colonies can make us better at communicating, decision making, and getting things done. Avery, New York 2010.
2. Benslimane D., Schahram D., Amit S.: Services Mashups: The New Generation of Web Applications. IEEE Internet Computing, Vol. 12, No. 5, 2008, p. 13÷15.
3. Amiri M., Shirgahi H.: Designing buyer and seller intelligent agents in an electronic market based on emergency decision making. International Journal of the Physical Sciences, Vol. 6(6), 2011, p. 1244÷1248.
4. Parunak H., Brueckner S. A., Sauter J. A., Matthews R.: Global convergence of local agent behaviors. Proc. Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, 2005.
5. Waligóra I., Małysiak-Mrozek B., Mrozek D.: Uniwersalna platforma wieloagentowa UMAP. Studia Informatica, Vol. 32, No. 2B(97), Gliwice 2011.
6. Nagios home page, http://www.nagios.org/ (DOA: 2.01.2012).
7. JADE (DOA: 15.01.2012).
8. Nagios Core Object Definitions Documentation, http://nagios.sourceforge.net/docs/nagioscore/3/en/objectdefinitions.html (DOA: 2.01.2012).
9. Nagios Core Custom Object Variables, http://nagios.sourceforge.net/docs/3_0/customobjectvars.html (DOA: 2.01.2012).
10. Nagios Plugin API, http://nagios.sourceforge.net/docs/3_0/pluginapi.html (DOA: 2.01.2012)
11. Nagios Plugin Development Guidelines, http://nagiosplug.sourceforge.net/develo-per-guidelines.html (DOA: 2.01.2012).
12. Nagios Core External Commands Documentation, http://nagios.sourceforge.net/docs/nagioscore/3/en/extcommands.html (DOA: 2.01.2012).

13. Nagios Core External Commands Register, http://old.nagios.org/developerinfo/externalcommands/commandlist.php (DOA: 2.01.2012).
14. Nagios Core Main Configuration File Options Documentation, http://nagios.sourceforge.net/docs/nagioscore/3/en/configmain.html (DOA: 2.01.2012).
15. Nagios configuration files and parallel_check command, provided for solution verification, https://sites.google.com/site/flecabinet/downloads/ BDAS12a.zip?attredirects=0&d=1 (DOA: 2.01.2012).
16. Standard FIPA Specifications, http://www.fipa.org/repository/standardspecs.html (DOA: 15.01.2012).
17. AgentBuilder Reference Manual Version 1.4 Rev. 0, by Acronymics, Inc., http://www.agentbuilder.com/Documentation/ReferenceManual-v1.4.pdf (DOA: 15.01.2012).
18. NetLogo Interface Guide, http://ccl.northwestern.edu/netlogo/docs/ (DOA: 15.01.2012).
19. Documentation Set for Swarm 2.2, http://www.swarm.org/swarmdocs-2.2/set/swarm.overview.mag3.observer-swarm.sect1.html (DOA: 15.01.2012).
20. IBM Tivoli Monitoring: Administrator's Guide, http://publib.boulder.ibm.com/infocenter/tivihelp/v42r1/index.jsp?topic=%2Fcom.ibm.omegamon.share.doc%2Fzconfig-com-mon08.htm (DOA: 15.01.2012).
21. IBM Tivoli Netcool/OMNIbus Administrator's Guide (former Micromuse Netcool/OMNIbus), http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIbus.doc_7.3.0/web_pdf_adm_master_73.pdf (DOA: 15.01.2012).
22. BMC Event Manager documentation, https://communities.bmc.com/communities/BEM (DOA: 15.01.2012).
23. HP OpenView documentation portal, http://www.openview.hp.com (DOA: 15.01.2012).
24. Network Monitoring VPS with Nagios, OpenNMS, Zenoss and More. Blog entry by Chinonye, July 5, 2011, http://myhosting.com/blog/2011/07/network-monitoring-vps-nagios-opennms-zenoss-more/ (DOA: 15.01.2012).

Wpłynęło do Redakcji 14 stycznia 2012 r.

Omówienie

Artykuł przedstawia przystosowanie bezpłatnego oprogramowania Nagios do monitorowania systemów agentowych. Nagios przeznaczone jest do monitorowania kluczowych usług i elementów infrastruktury w środowiskach IT. Ich cechą jest znacząca statyczność: te same

usługi świadczone są na tych samych elementach infrastruktury przez cały czas. Dodatkowo Nagios wymaga powiązania każdej definiowanej usługi z konkretnym hostem, na którym jest świadczona (Service A oraz Host na rys. 1).

Tymczasem w systemach agentowych liczba agentów jest płynna, ich kod może być wykonywany na różnych węzłach obliczeniowych, poszczególne agenty zaś mogą realizować zmienne role w trakcie rozwiązania problemu. Nagios oferuje jednak bogate możliwości konfiguracji oraz wyraźnie oddziela część zasadniczą (Nagios Core) od prostych zewnętrznych programów (wtyczek), które można rozwijać oddzielne i wykorzystywać na potrzeby Nagios dzięki stosowaniu publicznych interfejsów. Monitorowanie polega na cyklicznym wywoływaniu zewnętrznego programu i interpretowaniu zwracanych przez niego wartości jako statusu monitorowanego elementu. Każdy element jest definiowany oddzielnie, potencjalnie z podaniem własnych sposobów monitorowania.

Przyjęte rozwiązanie polega na rozszerzeniu Nagios w części dotyczącej jego wtyczek o jedną komendę (*parallel_check*), która pozwoli wielokrotnie – zamiast jednokrotnie – wykonać zdefiniowany dla elementu program testujący. Dodatkowo wykorzystuje się przy tym fakt, że pole *address* w definicji hosta nie jest interpretowane, a jedynie przechowywane i udostępniane przez Nagios. Wymienione elementy pozwalają na skonstruowanie warstwy hosta wirtualnego, która pozwala uwolnić monitorowane elementy od powiązania z konkretnymi węzłami obliczeniowymi.

Wykorzystując dodatkowe, standardowe elementy Nagios Core (komendy zewnętrzne i ewentualnie dodatkowo komendy do analizy wyników programów testujących) oraz przenosząc listę identyfikatorów hostów do zmiennej użytkownika, można uzyskać możliwość dynamicznej aktualizacji informacji o hoście wirtualnym (rys. 2). Ponieważ informacje o nim można aktualizować na podstawie wyniku monitorowania innego hosta wirtualnego, możliwe jest tworzenie ciągów takich hostów, z których każdy kolejny obejmuje podzbiór identyfikatorów poprzedniego. Rozwiązanie pozwala jako identyfikatory hostów stosować nie tylko adresy IP, ale również jakiekolwiek identyfikatory, na przykład instancje wirtualnej maszyny Java czy numery procesów w systemie operacyjnym.

Address

Artur Opaliński: Politechnika Gdańska, Wydział Elektrotechniki i Automatyki, ul. Gabriela Narutowicza 11/12, 80-233 Gdańsk, Polska, Artur.Opalinski@pg.gda.pl.