Computer Science • Vol. 11 • 2010

Włodzimierz Funika[*, **], Mateusz Kupisz[*], Paweł Koperek[*]

TOWARDS AUTONOMIC SEMANTIC-BASED MANAGEMENT OF DISTRIBUTED APPLICATIONS

In this paper we present our approach to the management of distributed systems based on semantic description of available resources. We use ontologies for a semantic description of the monitored system and other aspects of monitoring and management (such as metrics) and introduce a feedback loop on underlying infrastructure. Such an approach allows to automate monitoring and the ease the work of administrator. We introduce concepts behind a novel automatic management system, SAMM, developed within our research. We discuss the core mechanisms used in the system – the estimation of future measurements, approach to knowledge gathering, and the process of decision making. Then we provide some details on the architecture and implementation of SAMM.

Keywords: *autonomicity, monitoring, management, ontology, Eucalyptus*

W KIERUNKU AUTONOMICZNEGO, OPIERAJĄCEGO SIĘ NA OPISIE SEMANTYCZNYM, SYSTEMU ZARZĄDZANIA APLIKACJAMI ROZPROSZONYMI

Publikacja ta przedstawia nowe podejście do zagadnień monitorowania i zarządzania systemami rozproszonymi, wykorzystujące ontologiczny opis zasobów przez nie udostępnianych. Podejście to wykorzystuje ontologie do opisu semantycznego monitorowanego systemu, a także innych aspektów monitorowania i zarządzania nim (np. dostępne metryki) oraz wprowadza sprzężenie zwrotne na monitorowanej infrastrukturze. Pozwala to na automatyzację procesu monitorowania i zarządzania w celu ułatwienia pracy administratora. Publikacja opisuje także działanie nowatorskiego systemu SAMM, który powstał w wyniku badań. Przedstawione zostały również koncepcje dotyczące estymacji pomiarów, tworzenia baz wiedzy oraz procesu podejmowania decyzji. Artykuł opisuje zarówno architekturę SAMM-a, jak i szczegóły implementacyjne.

Słowa kluczowe: *autonomiczność, monitorowanie, zarządzanie, ontologia, Eucalyptus*

1. Introduction

There are many tools which help in the management of distributed applications. They provide the power of controlling both hardware and software resources. The administrator's role is dual: on the one hand he/she has to find a configuration which will

[*] Department of Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, 30-059, Kraków, Poland, {funika,kupisz,koperek}@agh.edu.pl
[**] ACC CYFRONET AGH, ul. Nawojki 11, 30-950 Kraków, Poland, funika@agh.edu.pl

use resources optimally and on the other hand it is an administrator's responsibility to ensure that all SLA conditions are fulfilled. SLA specifies precisely which aspects of system are crucial for operation. Therefore terms of this agreement define which resource parameters should be observed. Performance monitoring and ensuring proper quality of service become more and more important nowadays, especially in widely discussed cloud systems [1].

Monitoring and management systems are usually dedicated to particular hardware and software environments. They provide fine-grained monitoring data and system-specific management options. This gives a high level of control and enables the usage of all available capabilities. Such an approach may be overwhelming at the beginning. The administrator has to gain knowledge about the architecture of underlying systems and how to use the monitoring and management software [4]. There is no tool which would be able to monitor and manage various systems and allows for comprehensive control of them.

Monitoring and management are very often loosely coupled. It is human who needs to analyse raw monitoring data and based on it to execute appropriate actions. Automating common activities would be a great help to the administrator. The problem is that in most cases the data supplied by a monitoring system lack additional information about their actual meaning, i.e. semantics in the context of system operation. Such an additional description might be used for automatic processing and reasoning. There has been done some work concerning semantic monitoring. The work we are presenting in the paper is based on the results of research in this domain.

This paper is organized as follows: in Section 2 we discuss related research, next, in Section 4, we explain our concept of automation of management, and in Section 5 we present the architecture in depth. The next section focuses on implementation details. After that we briefly overview the implementation progress and then in section 7 our tests and their results. In the final section we discuss our conclusions and outline future work.

2. Related work

As previously mentioned there are a number of results of research in the area of automatic management, among others, based on semantics. Due to limited space we are concentrating on a constrained part of them.

In [8] a QoS-guarantees driven performance model is proposed. The model describes development, deployment and operations which allow to meet QoS and SLA constraints. An algorithm that allows to reduce computation costs at runtime is introduced. The constraints defined in form of SLA and QoS are also considered in the optimization process to avoid violation of agreements. Such an approach to management was tested on a cloud system, called CERAS running up to 1000 virtual machines and showed remarkable benefits in terms of their profit function gain and scalability. The main drawback of this on-going research is focusing on optimizing only one aspect of the system.

The [12] shows another approach to monitoring of SLA contract in distributed environment. Unfortunately, it is limited only to web services. SLA contract is formalized using of WSFL and BTP/ebXML notations. Monitoring is achieved by web service instrumentation. The gathered data is used in validation against contract conditions. Any violations are saved by the computing engine so that later on the administrator can tune underlying resources or the manager can renegotiate Service Level Agreements.

SemMon [5] project is aimed to create a tool providing semantic analysis of the monitoring data on a distributed system. Knowledge about effective tool usage (the relevance and accuracy of the metrics used during a system examination) may be shared within the research team with a built-in scoring system. All resources and metrics used are elements of a specific ontology [13] which provides a semantic description of gathered data. Thus SemMon is enabled to interpret data and can trigger indispensable actions, e.g. provide some suitable notifications (alarms) to users or launch additional, semantically close measurements. Its capabilities for integration with distributed computing environments were proven by integration with the ProActive framework [2] (please see [6, 7]).

IC2D [2] is an example of monitoring and management system dedicated to the ProActive framework. It is able to visualize the current state of computation by drawing activities (tasks) on every hardware or virtual node. Being dedicated to ProActive it allows for the monitoring of specific metrics like request queue length or response time. It also enables fine-grained management operations such as migrating computational tasks between nodes.

Oceano [3] is a prototype of infrastructure for a hosting center which is scalable and highly available. It is based on the observation that usually customers' systems experience loads which do not require a lot of resources for effective handling. Unfortunately, from time to time extremely high loads occur. Ensuring that such situations will be handled flawlessly requires reservation of big amount of resources which remain idle for most of the time. Of course this leads to unnecessary increase of system operation cost. Oceano mitigates this problem by sharing resources between customers who don't have big requirements in the particular moment and those who are facing high load peaks. Whenever high load occurs Oceano automatically "borrows" unused servers to guarantee that SLA conditions remain in tact.

3. Research objectives

In this paper we present our concept of autonomic management of distributed applications. It is based on a generic system description and feedback that our application can take on a monitored system. Our research is focused on exploiting the semantic description of resources and providing feedback in from of predefined actions from the monitoring facility to the managed system.

We use ontologies because they provide a universal medium for describing monitored system's resources, metrics that are monitored as well as the Service Level

Agreement that we want to fulfil. We also process monitoring data and give it semantic meaning. But our usage of ontologies goes beyond this. We use it also to describe actions that can be taken as a feedback on the monitored system so that an SLA contract would stay in tact. Since ontologies are easily extensible we are not tied to the underlying system w.r.t. neither metrics nor feedback actions that come as standard.

Automatic management requires a precise and consistent specification of system state evaluation methods. Without these information it would be impossible to determine if application is functioning properly and which actions should be executed. The SLA is a perfect source of such description. It contains specification of parameters considered crucial for successful operation and includes *Quality of Service* requirements which can be used to compare the state from different points of time.

With this approach automation of common administrative tasks may be achieved. Such software feature could bring significant cost reduction. Administrative effort could be decreased and the quality of management may be improved. The administrator would have to define the conditions under which the system is considered as operating properly and monitor the general situation in environment.

Advantages of such tool should be especially noticeable in cloud systems where one pay for every time unit when using each virtual machine. Depending on the current demand for processing power, thanks to our concept of feedback, virtual machines could be turned on only when needed.

4. Semantic-based Autonomic Monitoring and Management

In this section we provide an introduction into our concept of autonomic monitoring and management based on semantics. First, we discuss a general overview of main design assumptions and what requirements for this kind of tool were considered. Next, we come to the architecture of the system – its modules and their roles in proper functioning as a whole.

4.1. Concepts

The main problem addressed by this research is to help administrators to handle SLA/QoS and reduce costs of the exploited infrastructure. When working with the SLA/QoS parameters, the administrator has to perform following activities:

1. Recognize the requirements against the managed resources. The administrator needs to profoundly understand what terms does the SLA contain and their consequences. This step is necessary, e.g. to properly choose the resources which will be used to realize the contract.
2. Monitor infrastructure with various tools and metrics. The fulfilment of SLA has to be constantly monitored. Information about system utilization is required to determine how users should be charged for specific service usage (or to which of them any contractual penalties should be paid). It is also very helpful for the

service provider to know what is the current status of particular resources to determine which of them makes problems.

3. Execute some actions to optimize the usage of both hardware and software resources. When the system state is well known and the system administrator has the whole required knowledge about actions which can be executed, the usage of resources can be optimized. The service provider can e.g. migrate virtual machines with low CPU usage to a single computational node and turn off some physical machines. The cost of infrastructure usage would be lowered without violating SLA terms.
4. Gain and document the knowledge and experience about how the system functions. All the information about the system's current status, taken actions and their implications has to be documented for further usage. Actions may lead to some unexpected results and costs. A deep investigation of all conditions may be required to determine a real source of problems and to avoid repeating mistakes anew.

These activities may be automated:

1. Required information can be extracted from SLA in digital form. SLA may be used directly to determine what resources are in the scope of interest, how they should be monitored and how to evaluate costs.
2. Monitoring of resources can be automated by using one of existing systems. In our approach the observation capabilities are extended with the ability to interact with the resources under monitoring. Furthermore the observed system is constantly checked if it is in the proper state. If any SLA violation occurs, an alarm should be raised. The information contained in an alarm message should help to pinpoint resources that have got problems.
3. The total cost of hardware and software usage – including penalties for violation of SLA terms, should be periodically computed. Based on this, it may be decided to perform some actions to reduce the financial charges. It should even be possible to decide, that accepting some violations may actually be more cost-effective than executing actions which would prevent them.
4. All information about actions and and measurements is stored in a database – it may be used further to perform some deeper analysis.

The system could function in form of a closed loop:

1. The measurements are made on the managed system.
2. Values are used to determine the current system state.
3. Decisions about taking actions are made.
4. Actions are performed on the managed system.

In order to avoid creating the framework from scratch, thus to speedup the realization of the concept, one of the existing systems should be considered as a basis for research.

5. Architecture

Figure 1 depicts the architecture of the discussed system. The structure should be modular – created of loosely coupled components with well-defined interfaces. This would enable the components to be easily reused in other software projects.

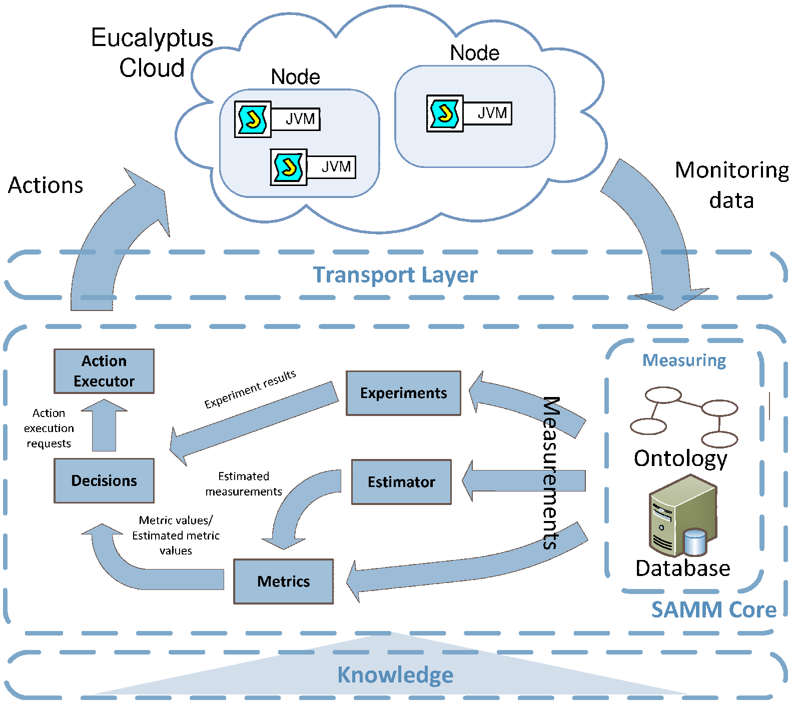

Fig. 1. Architecture of SAMM

The **Measuring** component gathers values from the registered resources according to their monitoring capabilities. The measurements are done when a specific metric value is assessed. Each measurement is done in the context of a specific resource of a well defined type, its capability to perform specific measurements and at a specific point in time. All this information is stored in database for further use in other modules e.g. in *Estimator*. Measurements may be taken using different technologies e.g. with JMX [14]. An abstract model of transport technology (the "transport layer") has to be prepared. To introduce a new way of measurements one should have to simply provide an implementation of a specific interface and, if necessary, prepare a Graphical User Interface to handle a new type of addressing. **Metrics** component computes the values of metrics enabling the evaluation of a current system state. Metrics are constantly being observed in case of any SLA violations. Each metric value is computed periodically on the basis of the most recent measurements. As mentioned above, required data is obtained from the Measuring component. The **Es-**

timator component provides estimations of future measurements values in a specific time frame. Estimations are used by the **Decisions** making module. The latter one calculates the total cost of system functioning according to the information provided by SLA and predicts a future level of resources usage with the estimated values of metrics. Such predictions are used then to rate the effects of all actions possible to execute on the managed system – including the case when no action is currently done. Then decisions may be made on the basis of the following questions:

- Whether any action should be currently carried out?
- If yes – which action?
- On which resource?

Once all the data about further management steps are collected, the execution of changes to the system state is delegated to the **Action Executor** module, which performs the requested actions. As mentioned above – to be able to manage the system, we need to know what are the implications of the actions taken. Such knowledge, at least to some basic extent, has to be collected before the current system operation. This is the responsibility of the **Experiments** component. It performs so called *experiments*: collects measurement values within a defined time frame before and after an action execution in a separate environment. With such data it discovers which measurement series were actually affected. This knowledge is then stored and used to decide what should be done at particular moments.

The **Knowledge** component is a common source of semantic information for other components. Types of resources, relationships between them, their capabilities, available monitoring metrics are stored within it in form of ontology. Thanks to flexibility of such kind of description, information regarding different aspects of system are bound to each other. Results of queries to such database include these relationships and are extended with some inferred facts added by automatic reasoner. Listing of all metrics which can be used to monitor a resource may be considered as a sample query to **Knowledge** component. Such query finds specified resource type, checks what measurements can be done on this resource and then looks for metrics which can be computed basing on provided information.

6. Implementation

We have developed a prototype Semantic-Based Automatic Monitoring and Management system – SAMM. It is based on the previously created semantic monitoring system – SemMon, and uses the same ontology-based approach for resources description. Actions are also described in form of an ontology. Therefore the set of available means of interaction of the administrator and system may be easily extended. SemMon is capable of interpreting data and provides some suitable notifications (alarms) to the user as well as can launch additional, semantically close measurements.

To extend the capabilities of SemMon, the latter was rewritten in form of OSGi [11] bundles. This greatly improved the internal architecture and made the development of new functions easier. Another improvement is the ability to change the ontology used for describing system resources and metrics. To use another ontology the administrator simply has to deploy another bundle providing an implementation of the service interface responsible for providing necessary knowledge.

The most important contribution in the presented research, which makes SAMM not only the successor of SemMon but a completely new and independent system, is the capability of *making decisions* how to interact with the managed system. During maintenance, the proper process of decision making on introducing changes to the system is crucial for its uninterrupted operation. The service provider has to answer three key questions:

1. Whether any action is needed to improve the system usage or to optimize maintenance costs?
2. What action has to be done?
3. Which resource has the action has to carried out on?

The key factor which helps in answering these questions is the *total cost of the functioning of the system*. We consider this cost as a sum of all charges put on service providers since the managed system starts its normal operation. A single charge element in this sum is defined as a multiplication of a *measurable value* indicating the level of usage of a specific resource (e.g. CPU utilization metric) and a *charge for one unit of resource* measured with that value. It is assumed that all costs are delivered to the system through SLA.

The presented definition of the total system operation was used to create an execution strategy of actions. The strategy answers addressed questions in the following way:

1. An action is executed only if it would lead to gain an increase in the cost smaller compared to a situation when no action would be executed. In case there are several actions which meet this criterion, an action with the smallest cost increase is chosen.
2. and 3. All the actions possible to execute on resources are evaluated – their impact on the system execution cost is computed. The pair action–resource with the best cost optimization is chosen.

The evaluation of impact on the system operation cost is based on the information gathered during the experiments phase as well as on the estimates of future measurements. The experiments phase is a time frame when SAMM gains knowledge about the managed system. It tries to execute actions and observes their impact on the environment. A single experiment consists of the following steps:

1. Recording measurements in a defined time frame before an action execution.
2. Execution of a particular action on a specified resource.
3. Recording measurements in a defined time frame after the action execution.
4. Discovery of which measurement series were actually affected by the action.

The estimation of future measurements exploits machine learning algorithms. At first, a model for specific data series is created with use of historical data from a specified period of time. When a prediction is needed, the most recent values are processed by the model and estimations are made. All models are periodically updated by recently gathered values. To ensure the highest possible quality of the implementations of machine learning algorithms, an existing framework – WEKA [15], with a big user-base and many successful deployments, has been chosen.

In the estimation process, some classification algorithms were used. From a wide range of different approaches, we selected those ones which could operate on continuous values space. As such we consider procedures which didn't require additional discretization of analysed data. Sequences of historical measurements values were cut into tuples of length N and passed as input to the algorithms. The result of processing – the classification, was used as a prediction of next measurements.

7. Results

7.1. Description of test environment

Our test scenario involves using SAMM to monitor and take actions on virtual machines running in private Eucalyptus cloud environment. The scenario itself reflects common real-life situation where the user deploys his/her web application to a Amazon EC2 Cloud service. The test should reveal prospective cost reduction of running virtual machines in EC2 environment without scarifying the response time of the application to the application user.

Since our hardware resources were limited the test server was a single machine with the specification as shown in Table 1 running Apache HTTPD server (as a load balancer) and all necessary Eucalyptus services.

Table 1
Server's specification

CPU	Intel Pentium Dual-Core E5300 @ 2.6GHz
Memory	4GB RAM
OS	Ubuntu 10.04 64-bit

We should note that running all Eucalyptus components (Cloud Controller, Walrus, Cluster Controller, Storage Controller, Node Controller) on a single machine is not a recommended setup and has its limitations (like the impossibility to configure virtual local area networks) however it should not affect our scenario anyway.

We have prepared a Eucalyptus image of Ubuntu 10.04 with Tomcat 6.0 installed with deployed mvc-basic web application (it is a sample application provided by SpringSource which presents basic capabilities of SpringMVC framework). Tomcat is configured in a way so that it automatically looks for other Tomcat instances running in the local network and builds up a cluster.

Apache HTTPD server runs on the same computer as Eucalyptus. It is used as a load balancer using *proxy_balancer* module. Balancer members use HTTP protocol to access web application content served by Tomcat instances. Balancing is realized by equally distributing HTTP request among all active balance members (Tomcat instances).

7.2. Test procedure description

The first step to allow SAMM to function is to perform the "Initial learning". To do so, we ran one virtual machine in Eucalyptus environment and put moderate load on it using Apache jMeter. The "moderate load" is such a load that won't overload ("kill") the machine but also will moderately stress it. We have found that emulating 15 clients (setting Apache jMeter to use 15 independent threads to test) puts such a load. This way, when SAMM tests what kind of consequences different actions bring (this is the main purpose to run "Initial learning"), it can find both positive and negative effects (higher or lower load) of them.

The test itself was to check what (if any) action SAMM will decide to undertake to reduce the cost of running the system. To do so, we have defined an SLA agreement as follows:

- Cost of running a virtual machine: 0.09 per hour.
- Cost of breaking Tomcat Response time QoS agreements: 10 per each measurement exceeding given threshold.

The metric used for second condition was measuring average Tomcat response time from last two measurements. By default our measurements are taken every 20 seconds. It means that whenever the average response time increases above a threshold of 1.0 (which means that the average time to handle the values from the last two measurements exceeded 1s) we would have to pay extra 10**metric_value*. We use such an "artificial" cost and metric to guarantee the QoS. The cost of running a virtual machine corresponds to the cost of running one in the Amazon EC2 environment.

To test SAMM's behaviour we emulated 35 clients constantly accessing our web application. When SAMM estimated that executing an action would bring a overall cost reduction over estimated cost without executing an action it undertook a "Start virtual machine" action which started up a new instance of virtual machine serving our web application.

7.3. Test results

Figure 2 shows the momentary costs of running the system once SAMM has executed the action and the momentary costs of running the system if SAMM wouldn't do it. As it can be seen in the graph the momentary cost after SAMM has started a second virtual machine was mostly keeping close to 0 (we only paid for running the machine and not the extra penalty for not complying to QoS requirements).

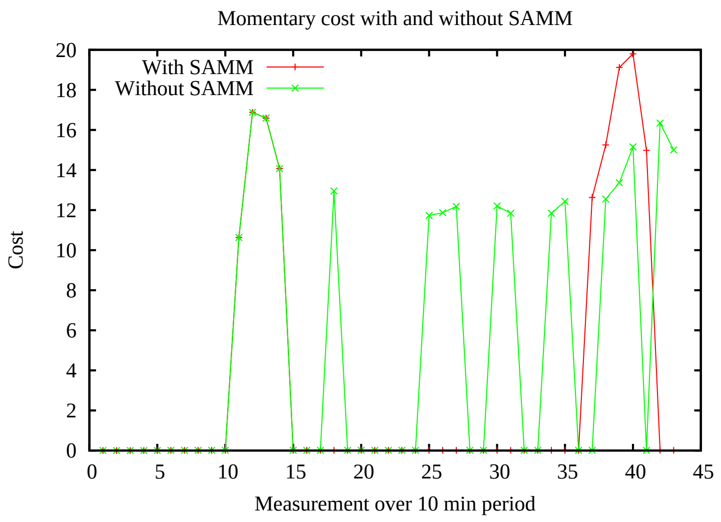

Fig. 2. Momentary cost

The overall (total) cost of running the system in the given 10 minutes time period would be:

- With SAMM: ∼140
- Without SAMM: ∼228

This, however, does not include the possible cost of executing the action itself. It is possible that the action execution may introduce an additional cost that would need some amortization time.

8. Conclusions and future work

SAMM is one of the first monitoring and management tools which aims to provide a integrated, automatic and customizable framework for distributed applications. Being based on its predecessor SemMon, it uses the semantics of monitoring data to validate QoS and SLA parameters in distributed environment. Estimating future metric values enables to predict possible contract violation and executes actions to prevent it. Actions can be also taken to optimize resources usage which may result in reasonable cost reduction. One of the main SAMM's features is that it is not dependent on the underlying system. Ontologies are used to describe resources, metrics, and actions, as well as the necessary quality parameters. This makes it easily extensible to support various distributed systems (in terms of resource monitoring) as well as to undertake system-specific actions.

Provided results show that using such tools may bring actual costs reduction. Although our testing scenario was not as complex as modern systems used in production environments, it proved that to some degree the decisioning process may be entrusted to the machine learning algorithms.

Our future work will focus on tuning the Decisions making module. Another direction for the system evolution is to introduce support for different time frames of estimation. We would like also to add support for more specific systems. As a first goal we would like to integrate support for gathering data from OCM compliant monitoring systems [10] as well as to add native support for collecting data from the NAGIOS system. [9].

Acknowledgements

The research presented in this paper has been partially supported by the European Union within the European Regional Development Fund program no. POIG.02.03.00-00-007/08-00 as part of the PL-Grid Project (www.plgrid.pl) and ACC Cyfronet AGH grant 500-08.

References

[1] Buyya R. et al.: *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility.* Future Generations of Computer Systems, vol. 25, 2009, pp. 599–616.

[2] Caromel D. et al.: *ProActive Parallel Suite.* `http://proactive.inria.fr/`, last accessed August 7, 2010.

[3] Fakhouri S., Fong L., Goldszmidt G., Kalantar M., Krishnakumar S., Pazel D. P., Pershing J., Rochwerger B.: *Oceano – SLA Based Management of a Computing Utility.* `http://www.scientificcommons.org/42509179`, 2001.

[4] Funika W. et al.: *Adapting a HEP Application for Running on the Grid.* Computing and Informatics, vol. 28, 2009, pp. 353–367.

[5] Funika W., Godowski P., Pegiel P.: *A Semantic-Oriented Platform for Performance Monitoring of Distributed Java Applications.* Proc. of International Conference on Computational Science, 2008.

[6] Funika W., Kupisz M., Koperek P.: *Integration of the SemMon semantic monitoring tool into the ProActive platform.* Proc. of Cracow Grid Workshop, Kraków, 2009, pp. 156–163.

[7] Funika W., Caromel D., Koperek P., Kupisz M.: *Integration of ProActive and the semantic-oriented tool SemMon.* Proc. of CoreGRID workshop, in conjunction with EuroPar 2009 Conference, Delft, The Netherlands, 2010.

[8] Li J., Chinneck J., Litoiu M., Iszlai G.: *Performance Model Driven QoS Guarantees and Optimization in Clouds.* Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009.

[9] *NAGIOS website* `http://www.nagios.org`, last accessed August 12, 2010.

[10] *OCM-G website.* `http://grid.cyfronet.pl/ocmg/`, last accessed August 5, 2010.

[11] *OSGi website* `http://www.osgi.org`, last accessed August 10, 2010.

[12] Sahai A., Machiraju V., Sayal M., Jie Jin L., Casati F.: *Automated SLA Monitoring for Web Services.* Proc. of IEEE/IFIP DSOM, 2002, pp. 28–41.

[13] Bechhofer S. et al.: *W3C, OWL Web Ontology Language Reference.* `http://www.w3.org/TR/owl-ref/`, last accessed August 3, 2010.

[14] Sun Microsystems, Inc.: *Java Management Extensions (JMX) Technology.* `http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement`, last accessed July 22, 2010.

[15] *WEKA framework website* `www.cs.waikato.ac.nz/ml/weka/`, last accessed July 25, 2010.