Computing and Informatics, Vol. 31, 2012, 427–446

SEMANTIC-ORIENTED PERFORMANCE MONITORING OF DISTRIBUTED APPLICATIONS

Włodzimierz Funika, Piotr Godowski
Piotr Pęgiel, Dariusz Król

AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Electronics
Department of Computer Science al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: {funika, dkrol}@agh.edu.pl

Communicated by Jacek Kitowski

Abstract. Monitoring services are an essential component of large-scale computing infrastructures due to providing information which can be used by humans as well as applications to closely follow the progress of computations, to evaluate the performance of ongoing computing, etc. However, the users are usually left alone with performance measurements as to the interpreting and detecting of execution flaws. In this paper we present an approach to the performance monitoring of distributed applications based on semantic information about the monitored objects involved in the application execution. This allows to automate the guidance on what to measure further to come to a source of performance flaws as well to enable reacting on interesting events, e.g. on exceeding SLA parameters. Our research comprises the implementation of a robust system with semantics, which is not biased to an underlying "physical" monitoring system, giving the end user the power of intelligent monitoring functionality as well as the independence of the heterogeneity of distributed infrastructures.

Keywords: Monitoring, knowledge, ontology, collaborative tools, JMX, SOA, SemMon

1 INTRODUCTION

The design of distributed applications is in many cases a challenge to the developer [1, 2]. On the one hand, there are the limitations and performance issues

of distributed programming platforms. So one of the most important tasks is to increase the performance and reliability of distributed applications. On the other hand, the developer must assure that the application manages and uses distributed resources efficiently. Therefore, understanding application's behaviour through performance analysis and visualization is crucial. Applications developed using modern technologies are difficult to monitor, due to the existence of different, technology-related factors which distract the developer from focusing on the application business logic. Thus, some kind of an abstraction layer on top of *low level* metrics is highly desirable.

When using performance tools (especially those working "on-line") the users are facing their complexity. Thus many users often benefit from less complex but easier to use tools. So a very important task is to ease user's interactions with the monitoring system, and moreover, to turn these interactions into a kind of collaboration with the system, which involves other users. Nowadays, more and more developed software tools provide a functionality that *guides* the user step-by-step. It can be implemented using a semantic description of software's features and through the analysis of user's behaviour along with the already existing observations to provide suggestions what to do to achieve a desired result, e.g. in the understanding of application's behaviour, flaws or malfinction.

The *primary goal* of our paper is to present an approach to application monitoring based on knowledge, which can cause a significant increase of a user's positive experience from the one hand and a decrease of an effort related to finding weaknesses/flaws (in terms of low performance) of an application from the other side. This goal is going to be achieved by exploiting the Semantic Web paradigm which has introduced the concept of semantic description of resources (OWL/RDF, DAML) and services (mostly Web Services – OWL-S, DAML-S).

The *second goal* is to address the issues stemming from the concept of leveraging the existing solutions to monitoring data acquisition, to develop a new layer of abstraction which would be oriented to the business logic aspects of monitoring process. Summing up, our approach is aimed to automate the detection of problems related to the malfunction of applications as well as performance issues. The novelty of our approach is multi-fold:

- the correlation of measurements coming from multiple distributed resources
- the development of a semantic model of relationships among the metrics assigned to different types of applications
- algorithms for adapting the measurements to the current state of the computation
- a model for the information to be collected on the execution of applications (including the latency, and useful lifespan)
- embedding the previous contributions within software agents, so as to address truly distributed scenarios.

In addition, we focus on an extensible platform whose aim is to enable connecting new systems, written by external developers to the developed one as it is rather difficult to foresee all the possibilities and context of tool re-use scenarios.

The rest of this paper is organized as follows: Section 2 gives a motivation and system requirements. Related work is discussed in Section 3. In Section 4 we present our proposed ontology and system architecture for on-line monitoring system with semantics, while Section 5 shows possibilities for extensions to the tool, followed by Summary and Future work in Section 6.

2 SYSTEM REQUIREMENTS

The following general use cases show the challenges to be addressed in performance monitoring from the user's perspective, should it be an end-user or administrator. The *end-user* like developer may be interested in such functions as:

- monitoring the performance of an application running under control of a *physical* monitoring system
- using the system in an automatic way with a set of metrics which are meaningful for the user and a desired result
- geetting information about metrics that should be called in a next step.

The *system administrator*'s concern relates to the operation of a system they have under control:

- create, destroy, and insert a semantic description of available metrics and elements of the monitored system
- provide new metrics in a *physical* monitoring system, and describe them in semantic way
- manage historical performance data.

Below we are presenting a semantic-oriented approach to the application monitoring process along with an implementation of the approach in form of an extensible monitoring platform called *SemMon*, which provides a semantic-based integration layer for so called *physical* or *low level* monitoring systems. These systems may be oriented to a concrete technology (e.g. JMX[1], or J-OCM [6]) and are required to provide monitoring data about the application, thus the SemMon can be used to monitor various technologies and applications in the same way.

By introducing semantics into the monitoring process, the system enables "understanding" what is really monitored, which in turn reduces the time the user spends to manually search for issues and shortens the system learning curve. Having a semantic description and taxonomy of the monitored elements and their contexts, the system is "smart" enough to guide its user throughout the whole monitoring/analysis process. The user can focus on its main task: to find performance

[1] Stands for *Java Management Extensions*

issues within limited time, based on the system guidance coming from historic analysis and being able to add their own measurements when needed.

The developed system should be designed in such a way that it should work with any existing "native" grid-enabled monitoring system. The system is targeted to be capable of integrating with existing ontology describing resources and performance measurements, which can be a great benefit for system administrators. The designed system should be able to be extended with sensors and metrics strongly related to the structure of the monitored application to point the actual and the most accurate source of the data.

3 RELATED WORK

In this section we concentrate on those available monitoring systems where semantics or flexible monitoring architecture are introduced.

Gemini [3] is a Grid monitoring framework developed within the K-Wf Grid project [4], which fulfils a gap between resources monitoring components and monitoring services clients. It performs measurements on workflows using a set of loadable modules called sensors which retrieve monitoring data on its own or by using external applications for this purpose. Although the Gemini framework is powerful in its flexibility of adding *new sensors*, it does not use any kind of semantics for selecting performance metrics to run and analyse or for providing any guidance to the user.

Autopilot [8] has been developed within the Grid Application Development Software (GrADS) Project [5] and is responsible for adaptive control of distributed applications. Autopilot's architecture comprises performance sensors and a decision control unit using fuzzy logic to analyse received data from sensors and preparing messages to actuators. Autopilot is the very first example of exploiting some kind of semantics usage, or rather *fuzzy logic* usage to help with monitoring and adaptation actions.

PerfOnto [9] is a new approach to performance analysis, data sharing and tools integration in Grids that is based on ontology. PerfOnto is an OWL ontology describing experiment-related and resource-related *concepts*. The experiment-related concept describes experiments and their associated performance data on applications. The prototype OPAS/PerfOnto system is able to search data in an ontological (i.e. using a knowledge base) manner, e.g. to find a code region executed on a particular node with a metric exceeding a threshold value, thus giving a hint to the site scheduler to migrate a job to another node. PerfOnto gives a rich description of performance data, but does not provide any automation for using it. Whereas using much of PerfOnto's taxonomy and retaining the main idea of describing resources in form of ontology, we aimed to significantly extend it and provide adaptation algorithms.

4 SEMMON SYSTEM VS. ONTOLOGY

The visual analysis of performance data in a "user friendly" form is one of the most key features provided by any performance monitoring system. Due to the great amount of gathered information, proper presentation and interpretation of observation results becomes a very complicated task. So steering the visualization of monitoring data involving making decisions on what, when, in what form, under which circumstances should be presented to the user is a challenge.

An overview of SemMon components is depicted in Figure 1.

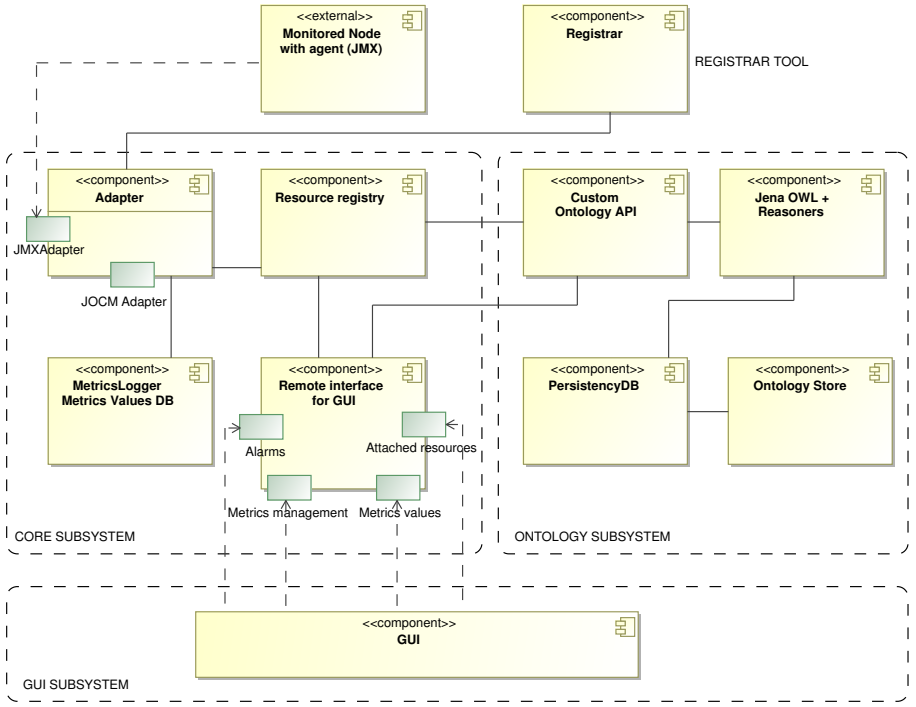


Fig. 1. Overview of system components

The focus of the model is the primary functionality aimed at processing ontology describing resources and their capabilities as well as metrics or storing monitoring data.

To support *knowledge persistency* a database is required. This functionality is implemented in a subsystem for handling ontologies (*Ontology subsystem*). Another focus of this node is support for a "physical" monitoring system. This subsystem has to provide a functionality for registering monitoring agents as well as for processing monitoring data.

The system contains computers with monitoring agents. The agents expose monitored resources to our monitoring system. All of them will register to the *Core subsystem*, afterwards Core is able to introspect possible resources that are exposed. Agents are programs on the *nodes/computers* that access the "physical" monitoring facility, e.g. OCM-G [12], JMX, J-OCM.

Access to SemMon is realized through *GUI clients* connected to the Core subsystem. Moreover, GUI is an environment for *collaborative work* – the users share metric ranks between different GUI instances in order to help other users in proper decision making. In the following we are focusing on a description of the components of the SemMon monitoring system.

The Ontology subsystem is the most crucial part of the whole system. The key aspect to understand here is the *ontology* term. An ontology is an explicit specification of a conceptualization. In such an ontology, definitions associate the names of entities (e.g., classes, relations, functions, or other objects) with a human readable text describing what the names are meant to denote, and formal axioms that constrain the interpretation and well formed use of these terms, formally specified with the OWL language. The Ontology Web Language (OWL) is intended to be used when the information needs to be processed in an automatic way by applications, as opposed to the situations where the content only needs to be presented to humans. OWL has powerful facilities for expressing meaning and semantics and thus OWL goes beyond all other similar languages in its capability to represent a machine interpretable content on the Web.

The Ontology subsystem contains methods for parsing, automatic interpretation, searching, creating, and, finally, saving and sharing ontology data. It brings a unique feature to the designed system: the capability of *interpreting what is monitored* both for system users and (what is even more important) for the system itself. Using the knowledge deployed in the underlying ontology data, the system is aware what is monitored and what should be monitored in a next step within the monitored application's lifetime. Every single type of resource accessible to the monitoring system is described in the OWL ontology and reflects a quasi-hierarchy structure (one resource can have more than one parent). Any part of the description or even the whole of it can be updated.

Resources in question are: *Resource classes* (like Node, CPU, JVM), *Resource instances* (i.e. OWL instances of resources available in the underlying monitoring system, like `CPU_i386_node2_cluster1`) and the *measurable attributes* for the resource instances. Each Resource class defines which measurable attributes are available for its instances. A measurable attribute, called *ResourceCapability* in this paper, might be both an atomic attribute (like `LoadAvg1Min`) or an OWL superclass for a set of *ResourceCapabilities*. This way a quasi-hierarchy of capabilities can be constructed. A special property `hasResourceCapability` is a glue between Resources and *ResourceCapabilities*. Any type of Resource can contain any number of Resource Capabilities. Figure 2 shows the Resources ontology class quasi-hierarchy while Figure 3 presents a fragment of the ResourceCapabilities ontology class quasi-hierarchy.

The metrics ontology describes metric concepts like OWL classes or individuals describing the metrics available to be executed by the user. It reflects a metrics quasi-hierarchy (i.e. from the most generic metric to the most specific one) in order to provide a rich description for ontology reasoners. The metrics can be *simple*, i.e. a metric is able to measure only one attribute, or *custom*, which means that the metric can be applied to as many capabilities as required and it is even possible to provide custom implementations for metrics (*user-defined metrics*).

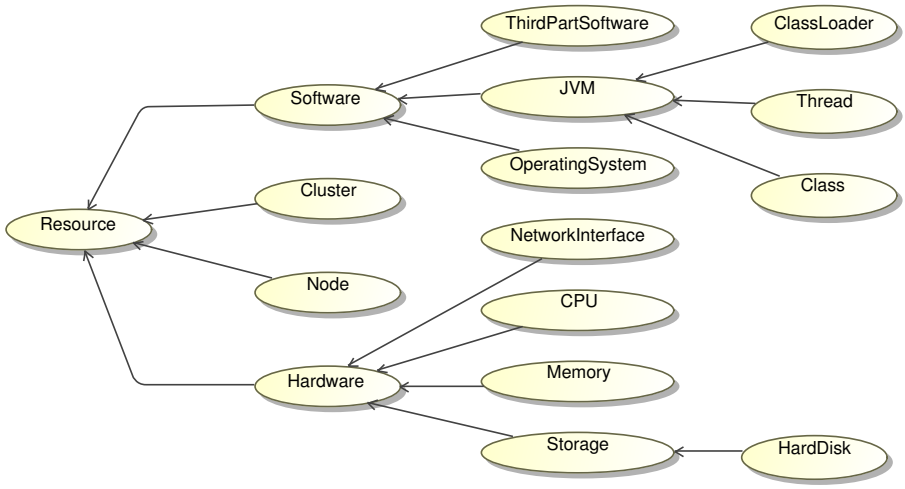


Fig. 2. Resources ontology diagram

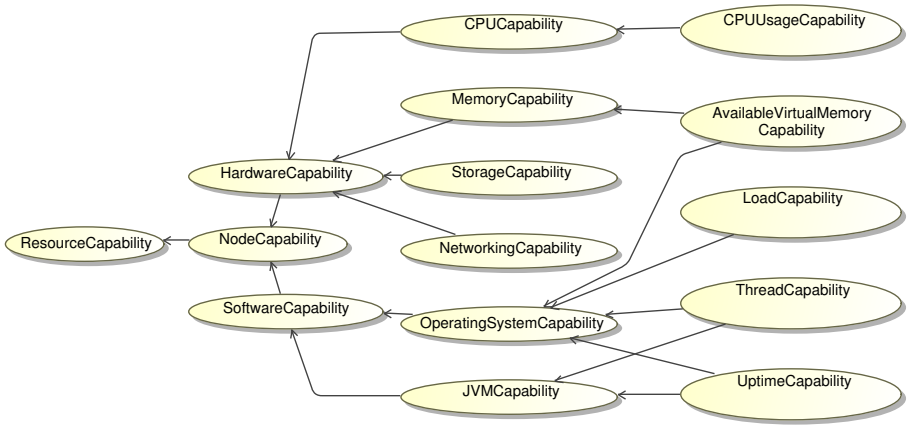


Fig. 3. ResourceCapabilities ontology diagram

The metrics ontology is derived from a flat list of all available metrics to be considered by the monitoring system. However, having only a flat list without a hierarchy (specialization) introduced, it is impossible to provide any powerful reasoning process. This is because no "`generic-specific`" or "`is related to`" relationships are provided. Looking at a flat list of all possible metrics, the next step is to find out which of them are *generic* and which are *specific*. Such relationships can be expressed in an ontology as the `rdfs:subClassOf` property. A sample superclass metric might be `SoftwareMetric` with its specific subclass `JVMThreadCPUTimeMetric`. As a result, metrics form a tree which can be used for a reasoning process. Figure 4 presents the Metrics ontology.

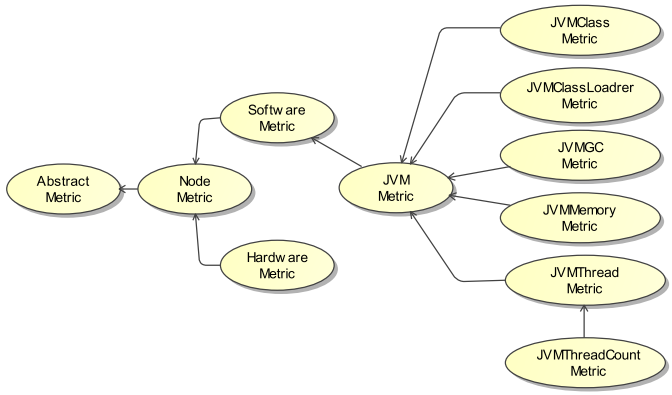


Fig. 4. Metrics ontology diagram

A special metric property `monitors` is a glue between the Metrics ontology and the Resources and ResourcesCapabilities ontology. It is used to express relations between metrics (*how to monitor*) and resources (*what can be monitored*). Property `monitors` has a domain in the `AbstractMetric` class (and its subclasses) and a range in the `ResourceCapability` classes (terms *range*,*domain* are defined as part of the OWL specification). Because the cardinality of this property is not limited, any type of `AbstractMetric` is able to monitor any number of capabilities. This means that the total number of measurements available in the system does not equal to the number of subclasses and individuals of the `AbstractMetric` class, but is a sum of cardinalities of the `monitors` properties in the metrics ontology.

The metric property `hasCustomImplementationClass` is used to inform the system that the metric is a *user-defined metric*, i.e. has its own implementation. This property points to the fully qualified Java class name implementing the `CustomMetric` interface. `CustomMetric` has its own implementation rules that is exactly returned as a measurement process, which is explained as follows. Since Custom Metric can access the Core public API, and the Resources registry, it can request any number of capabilities' values from the underlying monitoring system. The only

contract that a user-defined metric must meet is to return a single number each time it is requested for.

The metrics ontology contains also some additional properties (`DatatypeProperty`) which are used to store information about user interactions with the metric. All of the properties are sub-properties of the property `hasRankingProperty` which can be applied to any metric. The concrete properties are:

- `hasManualRunsCount` – describes how many times the user exploits this metric to investigate the system,
- `hasAverageUserRankValue` – the user is able to 'rank' the metric, a higher value means that the selected metric is more helpful than other ones (Section 5),
- `hasAutomaticRunsCount` – describes how many times the system automatically ran the metric.

The values of the properties are changing during user interactions with the system and are used to build the history of the metrics usage. This history, together with the automatic reasoning relations from the ontologies is used to create a list of metrics that can be run in the current state of the system (please refer to the sample usage in Section 6.1). The list of the metric is sorted by the weighted mean of those two indicators. The user is able to provide weights – so e.g. for some user the system could suggest the most frequently used metric instead of the metric that is 'semantically closest metric' (retrieved from the reasoning). The system is responsible for saving the properties in the ontology.

It should be noted that the described Ontology subsystem is able to coexist with any already existing ontology for resources description and matching.

5 HANDLING LOW-LEVEL MONITORING AND VISUALISATION

The Core subsystem is responsible for connecting to the underlying monitoring system's initialization (using its protocol adapter mechanism), deploying, initializing and executing metrics (including user-defined metrics), providing an interface to the Ontology subsystem and exposing a public (remote) interface for GUI clients to connect to. Core also manages GUI clients subscribed to the list of connected resources, running metrics, running metric values and alarms (i.e. conditional action metrics notifications). The Core subsystem comprises three components – *Adapter*, *Resource Registry*, and *Remote interface for GUI*. The *Adapter* component follows the commonly used Adapter structural design pattern and is used for "translation" of all Core requests into the requests specific to the underlying monitoring system (JMX, J-OCM, OCM-G, etc.). Due to the interface incompatibilities of a wide range of monitoring systems available on the market, a common interface called *Protocol Adapter* is designed. *Resource Registry* is a service that leverages both Core and Protocol Adapter. Resource Registry holds (with Protocol Adapter) all the resource instances found as visible in the underlying monitoring system and maps them into

Core identifiers. Protocol Adapter also resolves incompatibility issues between different physical monitoring systems. User-defined metrics have full access to the public Core API. Therefore a user-defined metric (implementing the *CustomMetric* interface) can introspect Resource Registry and with a Protocol Adapter implementation is capable to send a specific query to the underlying monitoring system. This feature is useful when the user wants to create a metric used to perform measurements on more than one resources (e.g. a metric that counts the number of the host in the network).

Remote interface for GUI allows remote GUI clients to connect to SemMon to enable collaborative work and provides:

- notifications for: the newly attached and detached *monitoring systems*, started and stopped *measurements* on the Core subsystem, and, finally, notifications for measurement values
- interface for alarms – Alarms are *conditional action metric* notifications. When some *action metric* is running on the Core subsystem and its value exceeds a user defined *threshold action value*, all the *unconditional action metrics* that are declared in the underlying ontology are sent as notifications to all the subscribed GUI users. The user is enabled to take an action to resolve the alarm (e.g. to start a new metric from within a list of metrics suggested by the system). Currently only one type of notification (alarm) is supported – the graphical alarm presentation in the GUI. In the future other notifications can be provided (like sending an e-mail, run a selected program).

One of the main objectives of GUI is to enhance performance visualization capabilities and to make system use much easier both for advanced users and for beginners.

The key GUI component in SemMon is the *Visualisation manager* that creates *visualisation* windows. This component allows the user to create, configure, show, and delete visualisations. It comprises two lists:

Running metrics list – the user can select which metric (started/running metric) he/she would like to add to the visualisation. The user is allowed to add more than one metric to one visualisation (please refer to the next paragraph for more details). By using this form the user has to be able to see an *average metric rank* (based on the rank from all users) and add an own rank. The rank is a number in range 1–5 and means how helpful the metric is for the user. Ranks are shared through all users and stored in the metric ontology for each metric.

List of visualisations – a list of the currently created visualisations. This list is private for the user – it is not shared with other users like the running measurements metrics list.

The *Visualisation* component is responsible for managing and displaying a single visualisation. The *visualisation* term is related to the presentation of data provided by the launched metric(s). The presentation layer uses different types of charts

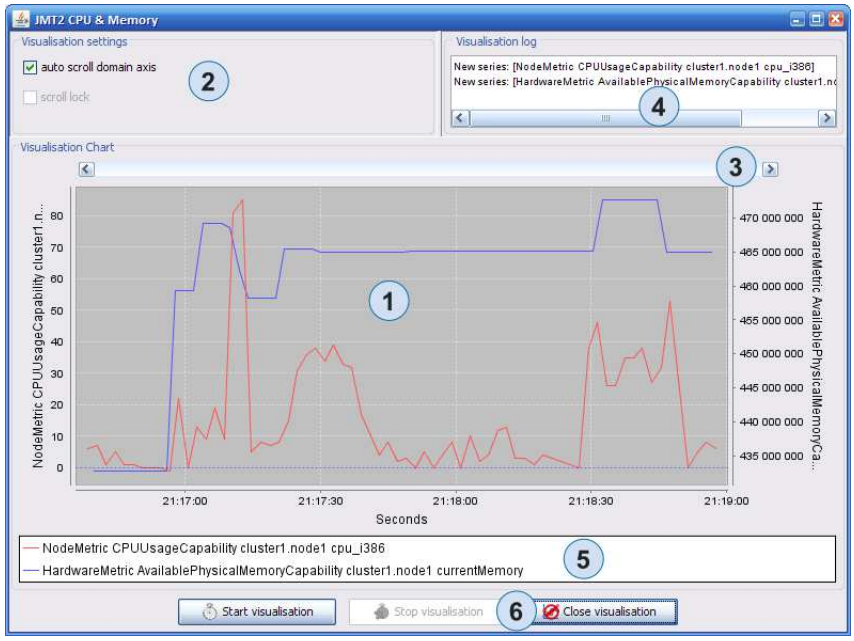


Fig. 5. Performance visualisation window

and options for these charts. A window with a sample visualisation is presented in Figure 5. This visualisation involves two metrics – the *CPU Usage* metric and *Available Physical Memory* metric. The visualisation is provided in panel **1**. This panel contains a chart based on some JFreeChart[2] components.

The main features of the chart are as follows:

- capability of dynamical creation of Y axis separately for different metrics on the same chart
- all the time (before visualization starts or afterwards) it is possible to add a new metric to the chart. Obviously, it possible to remove a metric, even after the visualization has been started;
- adding dynamically a new data to the chart (or updating it), the axis is automatically scaled (it is possible to control auto-scaling by using settings marked by **2**). If auto-scale is disabled the user can scroll the chart scroll box **3**.

Other functionalities supported by GUI include:

GUI Logic Center – it provides a universal model for many different system aspects, like: *Resource* structure, *Metric* structure, *Alarm notification* model.

[2] `http://www.jfree.org/jfreechart/`

GUI Alarm Manager – It is responsible for presenting to the user the actions that could be taken when an alarm occurs. It shows a list of available metrics to run in case of alarm.

Remote Adapter – It is a component which connects GUI with the Core subsystem. It also works as a façade that hides communication details from the components that store GUI's logic.

6 CASE STUDIES

In this section we explore a case of performance analysis of a Web application guided by SemMon, extending the system with a new functionality, and a study of the scalability of the system operation.

6.1 Performance Analysis of a Web Application

The below real-life example shows a few of the key SemMon features. A SemMon system user is monitoring a complex distributed application with critical problems relating to unstable memory usage over the application life time occurring only on a single node of the cluster. The memory usage is oscillating between 40 % and 99 % – the user can use SemMon to see the current state of the system. The monitored application is a WebService-enabled Java server, deployed across a cluster of processing nodes placed behind a restrictive firewall with a load balancer. The usual behaviour on the correctly deployed system in question is connected with consuming 50 % of the CPU time for each node and creating no more than 100 threads per JVM instance.

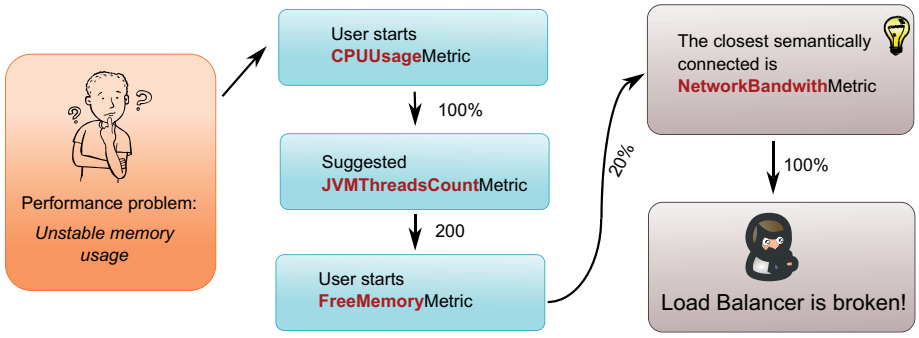


Fig. 6. Sample analysis with SemMon

At first, the user decides to monitor CPU usage on the heaviest loaded node (see step *1* in Figure 6). When a CPU burst lasts for at least 3 minutes, SemMon deduces a "critical" situation from the metrics ontology, and tries to build a list of the metrics suggested to start with. Since CPU load is semantically connected

with the number of JVM live threads (result of the metric ontology reasoning), the adequate metric is suggested and the user follows this guidance (*2*). Again, the number of threads is irrationally high (over 200 threads), so a new alarm is raised and SemMon "reasons" that since the CPU load and number of threads were already monitored, it would be reasonable to monitor memory usage (*3*). Since the memory usage is at the 80 % level of the available virtual memory, a new alarm is raised. This moment is the key in the described monitoring scenario: since the observed CPU load, memory usage, and JVM threads count are extremely high, the algorithm selects the *Network Bandwidth* metric to run (*4*). A motivation for doing this is that this metric had been frequently selected by other system users in the past and it is semantically connected (closest relation in the metric ontology) with memory usage, CPU load, and JVM threads (there is a possibility that application threads are processing data incoming from the network). SemMon has calculated the best matching metric and the user is able to see that almost the whole available network bandwidth is consumed by the incoming traffic. This suggests that either the cluster system is overloaded (which is not the case since we observe high load on a certain node only), or the load balancer is broken. The user checks the network bandwidth on the rest of the cluster nodes and does not observe any significant incoming traffic. This leads to the conclusion that the problem lies not in the monitored application, but in the load balancing component (*5*).

The analysis path followed by the user comprises both hardware (low level) and software (high level) metrics. It shows how flexible a reasoning process might be when the knowledge stored in SemMon holds possibly a full description of the environment. It is also possible to track down performance issues not only in the monitored application, but also in its environment.

6.2 Extending SemMon for SLA Monitoring

Below we focus on a case of integrating SemMon into the IT-SOA project[3]. The main goal of the project is to research novel methods and tools for practical application of the paradigm of Service Oriented Architecture (SOA) in the process of creating innovative solutions for improving the competitiveness of Polish enterprises, the development of the e-economy and information society. The SemMon platform is applied to monitor Service Level Agreements which describe the Quality of Service of partners within a Virtual Organisation. SemMon's features, e.g. the notification mechanism and support for different, physical monitoring systems, are crucial when considering different types (in terms of technology which was exploited) of services exposed by different companies [14].

During the research in the IT-SOA project a new requirement arose. The monitoring of SOA-oriented applications requires dedicated tools depending on an SOA framework. In the project, the Enterprise Service Bus (ESB) platform is widely exploited. It allows to build scalable applications from loosely coupled components,

[3] Home Page of the IT-SOA project – `http://www.soa.edu.pl`

each one being connected to the bus which exposes a coherent environment for message routing between the connected elements but at the same time can be distributed across multiple nodes.

For this type of environment a dedicated monitoring approach has to be designed. The IT-SOA project develops a novel solution for this issue (for more details please see [10]). To connect this monitoring with the SemMon system, an adapter component, which transforms data generated by the ESB monitoring system to the SemMon model, had to be developed. The implementation of the adapter uses a database in which the ESB monitoring system stores data gained from the performed observations. The schema of the database is generic in terms of metrics which are monitored, thus it can be used to store all the information from the ESB monitoring system. Apart from the measurement data, the database contains information about the monitored resources and metrics which can be monitored. The adapter for the SemMon system is responsible for connecting to this database and retrieving the requested information, e.g. on available resources, or the current value of a particular metric. Then, the adapter component transforms the data from a database specific form (tuples) to the SemMon model (semantic-oriented objects).

The most important lesson learned from this case study is the fact that the SemMon platform can be extended with new adapters to new monitoring systems without changing the code of the original SemMon code. The only new element is an adapter component which is separated from the core SemMon functionality. The user is enabled to monitor new types of applications with the same tool and the already assimilated functionality, e.g. step-by-step guidance through the monitoring process and an orientation to the factors which are important from the developer viewpoint, which allows to concentrate on application business logic rather than on low level aspects of application execution.

6.3 Performance Tests

We have studied the scalability issue of SemMon's operation due to its importance in large distributed systems. The implementation work has revealed that the main bottleneck in the SemMon system is the ontology processing engine. The main reason for this is that the Jena Semantic Web Framework [11] algorithms are rather brute-force and only the additional caching layer added in Jena improves the performance of ontology processing issues to a small extent. However, the more data is stored in the ontology, the slower the ontology subsystem works. Therefore the prepared test scenario concentrates on the measurement how the growth of ontology data affects either Core or GUI responsiveness. The test is based on SemMon usage performed during regular monitoring. The measurements are targeted at the following indicators: *time spent by Core* in the `Resources`introspection, *time spent by the GUI* to re-built the `Resources` tree, *consumed network bandwidth* used during communication between the Core and GUI.

The following environment was used to measure the responsiveness of SemMon Core and its GUI:

- PC station (Intel Pentium D 3 GHz, 4 GB RAM) for the SemMon Core and database application
- PC station (Intel Pentium Celeron M 1.6 GHz, 2 GB RAM) for the SemMon GUI
- PC station (Intel Pentium Celeron 2 GHz, 2 GB RAM) running up to 5 different Java applications with enabled JMX connector.

All nodes were running the GNU/Linux operating system and were connected using 100 Mbit isolated LAN network. Each node was running the latest Sun JVM 1.6 available (`1.6.0_01`). Each node has also the Pellet reasoner running for local queries (i.e. either GUI or Core uses its own reasoners to minimize querying queues, network latency and query-to-XML serialization overhead). The described configuration was chosen after preliminary tests which had shown that the best configuration should have the database (containing all the ontology data) running on the same node as the SemMon Core application. It also showed that the performance of SemMon is not limited by the prepared LAN configuration or memory available on each node.

For the purposes of testing a specially prepared Java application was invoked with JMX connector enabled. At start-up this application creates 100 threads, half of them communicating with the rest of threads using synchronized queues to pass simple 1 024 bytes long random "messages" at a random interval. Such an application should be a good example of a loosely coupled application exchanging calculation results with its environment.

# Attached nodes	Statements	Introspection time [s]	GUI refresh time [s]
1	250	20	1
2	500	50	2
3	750	120	4
4	1 000	280	6

Table 1. SemMon test results for RDBMS ontology model

Table 1 shows test results for the described environment, while Table 2 shows results when all the ontology data is stored in memory (Core and GUI are embedded in the same application, no RMI involved). When presenting the measured results graphically (see Figure 7), it is obvious that the introspection time grows exponentially with the growth of the number of ontology statements, where each statement represents an ontology triple[4]. However, when the relational database backend is removed, allowing Jena to use its "optimized" in-memory model, processing time growth is not so that visible as when used with database.

[4] An RDF triple: *subject, predicate, object* – `http://www.w3.org/TR/rdf-concepts/#dfn-rdf-triple`

# Attached nodes	Statements	Introspection time [s]	GUI refresh time [s]
1	250	5	1
2	500	7	1
3	750	10	1
4	1 000	15	2

Table 2. SemMon test results for in-memory ontology model

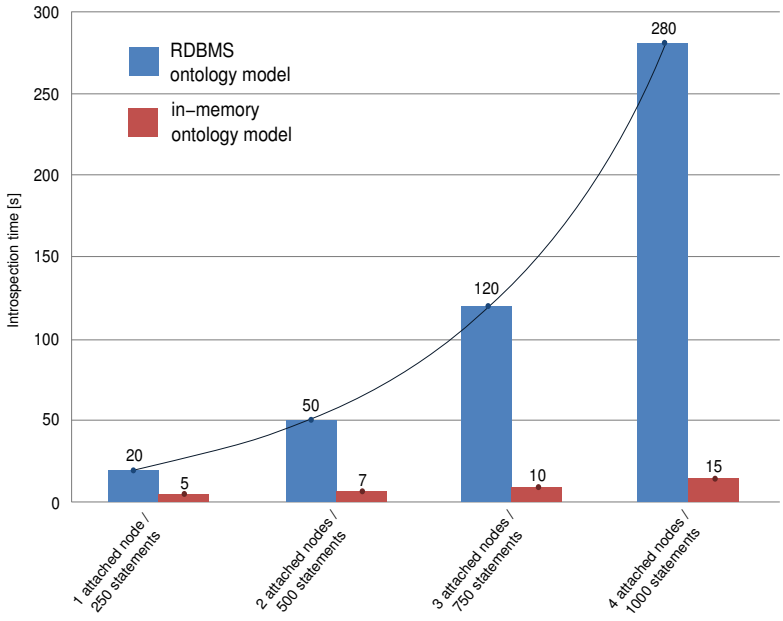


Fig. 7. Ontology processing times when using different ontology backend models

7 SUMMARY AND FUTURE WORK

The main objective of this paper was to present the design and implementation of a robust and flexible semantics-oriented monitoring system, SemMon. It seems to be one of the first complete approaches to the joint "worlds" of on-line distributed monitoring and Semantic Web.

The *SemMon* system extensively uses ontology for semantic description of all concepts used in. It is a flexible system, starting with picking up automatic ontology changes, through automatic metric selection assistance, collaborative users' knowledge leveraging, user-defined metrics, and finally, to extensible and clear visualisation options.

Due to the fact that *SemMon* is written in Java it can be run on every platform and operating system on which the Java Runtime can be found. We have

intensively tested SemMon on the following platforms: Linux/x86, Windows/x86, Windows/x64 and Solaris/SPARC. We used the SemMon system to monitor the operating system and Java based applications – all the metrics described in this article are ready to use.

There are still places for improvements. One of the unresolved issues concerns performing part of the computations on the clients to improve system scalability by reducing the size of performance data sent to the central database. An important task is to explore algorithms for reasoning in the ontology frameworks. Although there are some improvements in the query algorithms, they are just based on an additional caching layer rather than optimizing algorithms. Some of the new functionalities are described in next paragraphs.

Today's world is now focusing on the Internet and Web interfaces to applications. A Web interface could also be provided for monitoring systems. In our case we would still need the *Core subsystem*, but it might be possible to replace a heavyweight GUI client (which needs specialized software installed on the client machine) by a lightweight Web interface for performing a remote monitoring session. Using *Ajax*[5] and a modern Web framework (like *Ruby on Rails*[6] or *Symphony*[7]) the implementation of a Web interface should be relatively easy and quick.

The introduction of ontology and therefore a higher level of abstraction into the monitoring process enabled to overcome the problem of heterogeneity of the resources involved in computations and data storage [13]. There are further interesting domains which can be tackled based on the SemMon functionality: collaboration of humans and collaboration of tools. While the former kind of collaboration is quite popular in the research community, the second one, which concerns the ability of tools to co-operate with each other starting with making available some monitoring-related data up to synchronizing activities based on the notifying on coming actions, is less explored [15]. SemMon allows to introduce a high-level ontology-based protocols to enable co-operation between tools.

Integration with the K-Wf Grid *Grid Organizational Memory* (GOM) is also an interesting topic. GOM, a framework for knowledge deployment and management, uses the Jena Semantic Web Framework, too, but its design is more generic. It could be used as a *knowledge centre* which enhances the current *Ontology subsystem* with such features like automatic notifications about ontology changes, switching between different ontology management frameworks, etc.

In the next version of the SemMon system, dynamic instrumentation in the JMX Adapter will be supported. Using the dynamic JMX MBean loading service *MLet*[8], it will be possible to dynamically *instrument* a monitored application when monitoring with JMX. JMX Adapter could be enhanced by providing a set of cus-

[5] Asynchronous JavaScript and XML, a flavour of the XML-RPC protocol

[6] `http://www.rubyonrails.org/`

[7] `http://www.symfony-project.com/`

[8] `http://java.sun.com/j2ee/1.4/docs/api/javax/management/loading/MLet.html`

tom MBeans (*sensors*) (e.g. network throughput and network latency MBeans) and dynamically injecting them into the monitored application.

Acknowledgements

We are very grateful to Professor Jacek Kitowski and our colleague Paweł Koperek for valuable discussions. The research is partially supported by the POIG.01.03.01-00-008/08 "IT-SOA" project. Dariusz Król thanks to the UDA-POKL.04.01.01.01-00-367/08 project.

REFERENCES

[1] Gerndt, M.—Wismüller, R.—Balaton, Z.—Gombás, G.—Kacsuk, P.—Németh, Zs.—Podhorszki, N.—Truong, H.-L.—Fahringer, T.—Bubak, M.—Laure, E.—Margalef, T.: Performance Tools for the Grid: State of the Art and Future. APART-2 Working Group, Research Report Series, Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM), Technische Universität München, Vol. 30, Shaker Verlag, ISBN 3-8322-2413-0, 2004.

[2] Marco, J. et al.: The Interactive European GRID: Project Objecives and Achievements. In: Computing and Informatics, Vol. 27, 2008, No. 2, pp. 161–171.

[3] Balis, B.—Bubak, M.—Labno, B.: GEMINI: Generic Monitoring Infrastructure for Grid Resources and Applications. In: M. Bubak and S. Unger (Eds.)., Proc. Cracow Grid Workshop 2006. The Knowledge-based Workow System for Grid Applications, pp. 60–73, ACC Cyfronet AGH (Poland) (2007).

[4] K-Wf Grid web site: `http://www.kwfgrid.eu/`.

[5] Berman, F.—Chien, A.—Cooper, K.—Dongarra, J.—Foster, I.—Johnsson, L.—Gannon, D.—Kennedy, K.—Kesselman, C.—Reed, D.—Torczon, L.—Wolski, R.: The GrAds Project: Software Support for High-Level Grid Application Development. Technical Report Rice COMPTR00-355, Rice University 2000.

[6] Funika, W.—Bubak, M.—Smętek, M.–Wismüller, R.: An OMIS-Based Approach to Monitoring Distributed Java Applications. In: Yuen Chung Kwong (Ed.): Annual Review of Scalable Computing, Vol. 6, Chapter 1. pp. 1–29, World Scientific Publishing Co. and Singapore University Press, Singapore 2004.

[7] Zielinski, K.—Jarzab, M.—Wieczorek, D.—Balos, K.: JIMS Extensions for Resource Monitoring and Management of Solaris 10. In: V. N. Alexandrov, G. Dick van Albada, P. M. A. Sloot, J. Dongarra (Eds.): Proc. ICCS 2006, Reading, UK, May 28–31, 2006, LNCS, Vol. 3994, pp. 1039–1046, Springer 2006.

[8] Ribler, R. L.—Vetter, J. S.—Simitci, H.—Reed, D. A.: Autopilot: Adaptive Control of Distributed Applications. In: Proc. 7th IEEE High-Performance Distributed Computing Conference 1998.

[9] Truong, H.-L.—Dustdar, S.—Fahringer, T.: Performance Metrics and Ontologies for Grid Workows. In: Future Generation Comp. Syst., Vol. 23, 2007, No. 6, pp. 760–772.

[10] Psiuk, M.: AOP-Based Monitoring Instrumentation of JBI-Compliant ESB. In: 2009 Congress on Services I, pp. 570–577.

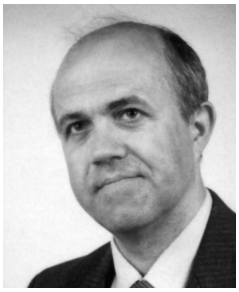
[11] Carroll, J. J.—Dickinson, I.—Dollin, C.—Reynolds, D.—Seaborne, A.—Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations. In: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, New York (NY, USA) 2004, pp. 74–83.

[12] Balis, B.—Bubak, M.—Funika, W.—Wismueller, R.—Radecki, M.—Szepieniec, T.—Arodz, T.—Kurdziel, M.: Grid Environment for On-Line Application Monitoring and Performance Analysis. In: Scientific Pogramming, Vol. 12, 2004, No. 4, pp. 239–251, `http://grid.cyfronet.pl/ocmg`.

[13] Kuna, D.—Jamro, E.—Russek, P.—Wiatr, K.: Using Standard Hardware Accelerators to Decrease Computation Times in Scientific Applications. In: Computer Science, Vol. 10, Annual of AGH-UST, Krakow 2009, pp. 65–74.

[14] Król, D.—Funika, W.—Slota, R.—Kitowski, J.: SLA-Oriented Semiautomatic Management of Data Storage and Applications in Distributed Environments. In: Computer Science, Vol. 11, Annual of AGH-UST, Krakow 2010 (to appear).

[15] Funika, W.—Janik, A.: Interoperability of Monitoring-related Tools. In: Computer Science, Vol. 7, Annual of AGH-UST, Krakow 2006, pp. 63-76.



Wlodzimierz Funika works at the Institute of Computer Science of the AGH University of Science and Technology in Krakow (Poland). His main research interests are in distributed programming, tools construction, performance analysis and visualization, machine learning. He is involved in EU Cross-Grid, CoreGRID, K-WfGrid, ViroLab, GREDIA, UrbanFlood projects.

Piotr Godowski graduated from the Institute of Computer Science of the AGH University of Science and Technology, Krakow, Poland in 2007. His main interests are in distributed systems, online monitoring, semantic analysis of performance related computing areas.

Piotr Pegiel received his M. Sc. in computer science at the AGH-UST in Krakow, Poland, in 2007. He is employed in Sabre Holding, participated in EU IST projects (ViroLab, Gredia). His interests refer to monitoring-related issues: knowledge based monitoring steering, self-healing of distributed systems, SOA/ESB/WS.

Dariusz Król received his M. Sc. in computer science at the AGH-UST in Krakow, Poland, in 2009. Now he is a Ph. D. student at the same university. His main research interests are in cloud computing, data storage management, performance monitoring and distributed systems. He is involved in EU IST ViroLab and GREDIA projects as well as in the Polish national PL-Grid project.