A semantic-oriented framework for the performance analysis of distributed Java applications

Włodzimierz Funika(1), Piotr Godowski(1), Piotr Pęgiel(1), Marian Bubak(1,2)
(1) Institute of Computer Science, AGH, ul. Mickiewicza 30, 30-059 Kraków, Poland
(2) ACC CYFRONET AGH, ul. Nawojki 11, 30-590 Kraków, Poland
{funika,bubak}@agh.edu.pl, {flash,pegiel}@student.agh.edu.pl
phone: (+48 12) 617 44 66, fax: (+48 12) 633 80 54

Abstract

In this paper, we present a new approach to the semantic performance analysis in the on-line monitoring systems. We have designed a novel monitoring framework which uses ontological description for all the concepts exploited in the monitoring of distributed systems. We discuss a complete implementation of a system with semantics, which is not constrained to any kind of the underlying “physical” monitoring system, providing the user with such features like automatic selection of metrics and collaborative work.

1 Introduction

In the world of nowadays computer science, developing new distributed applications has become a very complicated, difficult, often long-term process ([1, 2, 3]). On the one side, there are the limitations and performance issues stemming from the features of distributed programming platforms and one of the most important tasks is to increase the performance and reliability of distributed applications. On the other side, the developer must make the application to manage and use distributed resources efficiently. Therefore, understanding application’s behaviour through performance analysis and visualization is a crucial task. It is especially true now, when many distributed systems exploit the SOAP protocol [4] and the functionality of the program is divided into pieces and implemented as web services. The monitoring of data flow between components is very helpful for the user to find problems with the system involved.

When using performance tools (especially, those which operate “on-line”), one of the biggest problems the user faces is tools’ complexity. Thus many users give up what these systems offer and 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. Certainly, the simplification of the program use shall not lead to limiting its functionality related to performance evaluation. On the other hand, more and more developed software use software agents which guide the user step by step. Such agents usually use a semantic description of software’s features and through the analysis of user’s behaviour there emerge suggestions what he/she should do in order to achieve a desired result.

A similar approach can be used in tools used for performance monitoring in HPC. The first steps were already done (AutoPilot, PerfOnto), but the authors of these tools developed their own architecture for semantic description (mostly based on feedback from metrics), not using the already existing solutions (like OWL/RDF).

The Semantic Web paradigm has introduced the concept of semantic description of resources (OWL/RDF, DAML), semantic description of available services (mostly Web Services, OWL-S, DAML-S). It is possible to leverage from existing standards to develop a performance monitoring tool using some knowledge which describes performance metrics, thus providing insight into application’s behaviour and this is the primary goal of this paper.

The rest of this paper is organized as follows: Section 2 discusses a motivation and use cases for the system under discussion. Related work is discussed in Section 3, in Section 4 we present our vision of using ontology and system architecture for on-line monitoring system with semantics, followed by a discussion on the automation of deduction strategy in Section 5. A sample usage of the system is shown in Section 6. At the end we give Summary and Future work.

2 Motivation and system use cases

In this paper we focus on a semantic-oriented monitoring infrastructure called SemMon. The architecture of the
tool fits in the OMIS model [5] and is capable of working with available monitoring systems (like J–OCM[7], JMX). Semantics in the monitoring architecture should exploit as much as possible from existing solutions, libraries, and tools, with special attention paid to open source software and solutions developed in European projects (like Grid Organizational Memory (GOM) [8] developed in the K-Wf Grid project [6]). The following general use cases show what functions the designed semantic-oriented performance monitoring system should provide. The user is intended to be able to:

- monitor the performance of a Java application running under control of a physical monitoring system
- use in an automatic way a set of metrics which are meaningful for the user and a desired performance analysis result
- use a GUI tool for the visualization of performance data, browse available performance metrics, collect graphs of at least one metric in a single chart and check which available metrics are semantically connected with the current metric
- see information about metrics that should be called in the next step after using the first metric
- browse semantic descriptions of metrics in a user-friendly form.

The system administrator should be able to:

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

The work should be based on the XML format both for sending and receiving requests on semantic descriptions. Developing a system as a web service (with a semantic description provided) leads to a universal solution, which fits into the Semantic Web paradigm. The developed system should be designed in such a way that it should work not only with an OMIS–compliant system ("physical" monitoring system), but also with any existing monitoring solution for distributed systems.

3 Related work

In this section, related work concerning semantics usage in on-line monitoring for distributed systems is discussed. We are concentrating not on all available monitoring systems, but rather on those ones where semantics are involved.

Ganglia [15] is an open source monitoring system, deployed in over 500 clusters on the whole world. It was developed at the University of California in Berkeley, now it is maintained by the Planet Lab. Ganglia focuses on a hierarchical structure of clusters connected via IP network. It relies on a UDP multicast producer/consumer pattern to monitor states within clusters, while the TCP/IP protocol is used between clusters. Membership in a group of clusters is maintained by the reception of heartbeat messages sent at certain periods of time. With Ganglia it is possible to build a specialized tool that is able to interpret received data relatively easy. In addition, Ganglia provides highly-specialized algorithms to achieve very low overhead caused by monitoring and high concurrency.

Ganglia, however, does not use any kind of knowledge to help the user with the interpretation of monitoring data. It also is not able to adapt the monitoring process to a current situation on each cluster node.

Autopilot [9] was developed in the Grid Application Development Software (GrADS) Project [14] and is responsible for adaptive control of distributed applications. Autopilot provides the following features:

- Dynamic performance instrumentation
- Configurable, malleable resource management algorithms
- Based on application request patterns and observed performance
- Real–time adaptive control mechanisms.

Autopilot is mostly used to automatically configure resources. Its architecture consists of performance sensors, software actuators and 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.

Aksum [10] is a part of the Askalon [11] project, aiming to simplify the development and optimization of applications that can harness the power of grid computing. Aksum can automatically:

- instrument the user’s application
- collect the data generated by the instrumentation and analyze it
• relate performance problems back to the source code and compare the performance behaviour across multiple experiments

Aksum automatically searches for performance bottlenecks based on the concept of performance properties. In contrast to much existing work, performance properties are normalized (values between 0 for the best case and 1 for the worst case), enabling the user to interpret the resulting performance behaviour. In the Askalon project, the first ontology-based approach was invented. There was addressed the use of ontology for performance analysis domain. Ontology was developed for representing performance data on the Grid with the hope that the proposed ontology would not only serve for data sharing and reuse between performance analysis tools but also increase the automation of performance analysis process.

PerfOnto [12] is a new approach to performance analysis, data sharing and tools integration on grids, which is based on ontology. Basically, PerfOnto is an OWL ontology describing experiment-related and resource-related concepts. The experiment–related concept describes experiments and their associated performance data of applications. The structure of the concept is described as a set of definitions of classes and properties, like Application, Version, Code region and Experiment classes.

• Application class describes information about the application
• Version class describes information about versions of an application
• Source–File class describes the source file of a version
• CodeRegion class describes a static (instrumented) code region
• Code regions are classified into subclasses that are programming paradigm dependent and paradigm-independent
• Experiment class describes an experiment which refers to a sequential or parallel execution of a program
• RegionInstance class describes a region instance which is an execution of a static (instrumented) code region at runtime.

A region summary has performance metrics (property hasMetric) and subregion summaries (property hasChildRS). PerformanceMetric describes a performance metric; each metric has a name and value. In addition to the detailed description of performance resources and experiments, the prototype PerfOnto system is able to search data in an ontological (i.e., using knowledge base) manner. For example, it is possible to find a region summary executed on a particular node with a specified metric value exceeding some threshold value. Such queries can be used to monitor critical sections of executed application, triggering administration notifications or even can be a suggestion to a site scheduler to migrate an existing job to another node or to stop scheduling jobs for an overloaded node. PerfOnto gives a rich description of performance data, but does not give any automation for using it; neither there are adaptation algorithms.

4 Overview of the SemMon system

The visualization of monitoring data in an application–meaningful, "user friendly" form is one of the most key features provided by any monitoring system. Due to the great amount of gathered information, proper presentation and interpretation of observation results becomes a very hard and complicated task.

A high level architecture overview of the SemMon system is given in Figure 1.

![Figure 1. System architecture as a distributed environment. Monitoring core computer/cluster includes Core subsystem and Ontology subsystem.](image-url)
our monitoring system. All of them will register to the Core subsystem, after that Core will be able to introspect possible resources that are exposed. The agents are programs on the nodes/computers that use a "physical" monitoring system, e.g. JMX, J-OCM.

The last part of the system are computers with Graphical User Interfaces which are connected to the Core subsystem. All of the Core subsystem functionality are accessible from within GUI (e.g. browsing monitoring resources, running measurements) in a simple and user friendly way. It is optimally designed for the advanced user as well as for the beginners. GUI provides functionality for collecting some part of knowledge base data – like a metrics usefulness score. GUI is also an environment for collaborative work – users share metric ranks between different GUI instances in order to help others in proper decision making. In the following we focus on a description of the subsystems involved in the SemMon monitoring system.

4.1 Ontology subsystem

The Ontology subsystem is the heart of the whole designed system. It contains methods for parsing, automatic interpreting, searching, creating, and finally saving and sharing ontology data. The ontology subsystem brings a unique feature to the designed system: the capability to interpret 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 knows 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 natural computing resources hierarchy. There is a possibility to update part of or even the whole of the description.

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 resource instances. Each Resource class defines which measurable attributes are available for its instances. A measurable attribute, called in this paper ResourceCapability might be both an atomic attribute (like LoadAvg1Min) or an OWL superclass for a set of ResourceCapabilities. This way a natural hierarchy of capabilities can be constructed. A special Resource property hasResourceCapability is a glue between Resources and ResourceCapabilities. Any type of Resource can contain any number of ResourceCapabilities. Figure 2 shows the Resources ontology class hierarchy while Figure 3 presents a fragment of the ResourceCapabilities ontology class hierarchy.

An ontology describes metrics concepts like the OWL classes or individuals describing metrics available to be executed by the user. Metrics also conform to their hierarchy in order to provide a rich description for ontology reasoners. Metrics can be simple which means that the metric is able to measure only one attribute or custom which means that the metric can measure as many capabilities as required, it is even possible to provide custom implementation for metrics (user-defined metrics).

![Figure 2. Resources ontology diagram](image)

![Figure 3. ResourceCapabilities ontology diagram](image)

A metrics ontology is designed 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 JVMThreadCPUUtilMetric. As a result, metrics form a tree which can be used for a reasoning process.
A special metric property monitors is a glue between the Metrics ontology and the Resources and ResourcesCapabilities ontology. Property monitors has a domain in the AbstractMetric class (and its subclasses) and a range in the ResourceCapability classes. 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.

Metric property hasCustomImplementation-Class is used to inform the system that the metric is a custom metric, i.e. has its own implementation. This property points to the fully qualified Java class name implementing the CustomMetric interface. Custom Metric has its own implementation rules that is exactly returned as a measurement process, which is explained as follows. Since Custom Metric can access a Core public API, and the Resources registry, it can request any number of capabilities’ values from the underlying monitoring system. The only contract that Custom Metric must meet is to return a single number each time it is requested for.

4.2 Core subsystem

The Core subsystem is responsible for connection to the underlying monitoring system’s initialisation (using its protocol adapter mechanism), deploying, initialising, and executing metrics (including user–defined metrics), providing an interface to the Ontology subsystem and last but not least, Core has 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 consists of 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 major differences and 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 couples the Core and Protocol Adapter subsystems. Resource Registry holds all found (with help from Protocol Adapter) resource instances in the underlying monitoring system and maps them into Core identifiers. Protocol Adapter resolves incompatibility issues between different “physical” monitoring systems.

User–defined metrics have full access to a public Core API. Therefore a user–defined metric (implementing the Custom Metric interface) can introspect Resource Registry and using a Protocol Adapter implementation is capable to send a specific query to the underlying monitoring system. This feature is useful when the underlying monitoring system has some specific features, not covered by the generic Protocol Adapter interface.

Remote interface for GUI allows remote GUI clients to connect to SemMon to enable collaborative work. The following aspects of the system environment can be addressed by this interface:

- interface for attached resources – It is possible for a GUI client to subscribe to notifications about newly attached and detached monitoring systems and their resources,
- interface for running metrics – A GUI client is enabled to subscribe to notifications about newly started and stopped measurements running on the Core subsystem,
- interface for running metrics values – In addition to subscription to the running metrics notification, GUI clients can subscribe to selected metric values. After a successful subscription of a GUI client, metric values are sent to the client at the same time period as defined in the polling time interval of a running metric,
- interface for alarms – Alarms are conditional action metric notifications. When some action metric is running on the Core subsystem and its value exceeds a declared 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).

4.3 GUI subsystem

The visualization of data delivered from the monitoring system in a “user friendly” form is one of the most important issues for any monitoring systems. Due to the great amount of gathered information, the proper presentation, first of all, in order not to flood the user with a lot of relevant and irrelevant performance data, and interpretation of observation results, becomes a very hard and complicated challenge for tool developers. The user expects rather a kind of guidance, what and how, in what sequence to observe and act. Therefore one of the most important tasks for GUI is to allow for enhancements of performance visualization means and to make system use much easier and flexible, both for advanced users and for beginners.

The most important SemMon’s GUI component 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 visualisation. It is possible 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.

- List of visualisations – a list of the currently created visualisations. This list is private for the user – it is not shared with other users unlike a running measurements 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 and settings for these charts. A window with a sample visualisation is presented in Fig. 4.

![Figure 4. Performance visualisation window](image)

This visualisation comprises 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 JFreeChart\(^2\) components. The main features of the chart:

- possibility 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.

\(^2\)http://www.jfree.org/jfreechart/

- when dynamically adding a new data to the chart (or updating), the axis is automatically scaled (it is possible to control auto-scaling by using the settings marked by 2). If auto-scale is disabled the user can scroll the chart with scroll box 3.

The sample visualisation is only one of many features provided by GUI. Below we present a summary list of other facilities comprised in GUI:

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

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

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

## 5 Deductive strategy of automatic metrics selection

The deductive strategy used during the automatic selection of metrics heavily relies on the knowledge deployed in the Ontology subsystem and a current monitoring context. When an alarm occurs, e.g. when a metric exceeds some threshold value, SemMon has to decide which metrics should be taken into account in the next monitoring step. To achieve a reasonable goal, the following key properties among the other heuristic attributes are examined for each candidate metric:

- **specialization**: the metrics hierarchy is analysed using the subClassOf property to calculate the metric’s specialization

- **rank and historical data**: values associated with each metric and stored in the ontology origin based on the user ranks (resulting form their collaborative work) and statistical data about metric’s popularity in the past (thanks to knowledge persistency)

- **current monitoring context**: only metrics from semantically connected monitoring areas are taken into account (e.g. it does not make sense to monitor network bandwidth when a problem arises in the JVM garbage collector).

Using a weighted sum formula (weights are user-configurable), an overall metrics usefulness value is calculated based on the aforementioned attributes. These with the
up to 7 smallest values are suggested to the user as the best-guess selections. In case of unconditional action metrics, only the best "matched" metric is started. Surprisingly for the authors, thanks to the described best effort algorithm for the reasoning process the user is easily able to track down performance issues. Provided that SemMon’s GUI shows the history of user choices and current guidance, it should be always clear for the user how to interact with the system.

6 Sample use of SemMon

The below real–life example shows the main features of the SemMon system.

One of the SemMon system users is monitoring a complex distributed application with severe problems relating to unstable memory usage over the application life time occurring only on a single node of the cluster. The monitored application is a WebService–enabled Java server, deployed across a cluster of processing nodes hidden behind a firewall with an enabled load balancer.

Figure 5. Sample analysis with SemMon

At the first step, the user decides to monitor CPU usage on the heaviest loaded node. When a CPU burst lasts for at least 3 minutes, using its default metrics ontology SemMon recognizes a "critical" situation (since CPUMetric is an action metric, with a conditional alarm), and calculates the next best–fitted metric to start. Since CPU load is semantically connected with the number of JVM live threads, JVMThreadsCountMetric metric is suggested and the user follows this guidance. 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. Since the memory usage is at 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. A motivation for doing this is that NetworkBandwidthMetric was frequently selected by other system users and it is semantically connected with memory usage, CPU load, and JVM threads (it is possible that application threads are handling increasing network traffic). SemMon has calculated the best matching metric (i.e. NetworkBandwidthMetric) and the user is able to see that the network bandwidth is almost totally consumed by the incoming traffic. This suggests that either the cluster system is overloaded, or the load balancer does not work as expected. The user checks the network bandwidth on the rest of the cluster nodes and does not observe a significant incoming traffic on them. This suggests that the problem lies not in the monitored application, but in the load balancer software or the exploited load balancing strategy.

Please note that the path followed by the user comprises both hardware (low level) and software (high level) metrics. It shows how flexible a reasoning process should 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 (like in the above sample scenario).

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 system monitoring and Semantic Web.

The SemMon system extensively uses ontology for semantic description of all concepts used in. In addition, it is as much flexible as it can be, starting from picking up automatic ontology changes, through providing assistance in automatic metric selection, collaborative users’ knowledge leveraging, user–defined metrics, finally to the extensible and clear visualisation options.

There are still issues for further improvements. The main tasks for the future work are: switching to the pure SPARQL (instead of using Jena ARQ), using Java Web Start for the GUI, case study over AJAX based GUIs, and full integration with K-Wf’s GOM. One of the most important tasks is to develop algorithms for reasoning in the ontology frameworks. This is the key feature of the ontology and by now it still works in a brute–force scenario. Although there are some improvements in the query algorithms, they are just basing on additional caching layer rather than optimizing algorithms.

Hopefully, we are expecting that the future will bring a great progress in this area, since many scientists and commercial research centres (HP Semantic Labs3 and W3C Semantic Web Working Group4 are the perfect examples) are already hard working on the Semantic Web development.

3http://www.hpl.hp.com/semweb/
4http://www.w3.org/2001/sw/
Without the anticipated progress in the mentioned domains, the current version of the implemented system should be considered as an alpha version of productivity quality – it meets its functional requirements, but there are still performance-related issues. There are left some open issues to solve, but a robust, flexible, and extensible engine is ready for use in future research.

Acknowledgements. The research is partially supported by the EU IST 031857 int.eu.grid project.

References


[13] R. Johnson, E. Gamma, R. Helm, J. Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995
