

Instrumenting and Monitoring the LarKC Reasearch Infrastructure

Reto KRUMMENACHER¹, Ioan TOMA^{1,2}, Dieter FENSEL¹,
Raluca BREHAR³, Sergiu NEDEVSCI³

¹*Semantic Technology Institute, University of Innsbruck, Technikerstr. 21a,
A-6020 Innsbruck, Austria*

Tel: +43 (0)5125076485, Fax: +43 (0)5125079872, Email: first.last@sti2.at

²*Softgress S.R.L., Str. Plugariilor 20, Gherla, 405300, Romania*

Tel: +40 (0) 767684924, Email: first.last@softgress.com

³*Universitatea Tehnica Cluj-Napoca, Str. Gh. Baritiu 28, Cluj-Napoca, 400027, Romania*

Tel: +40 (0) 264 401219, Fax: + 40 (0) 264 594491, Email: first.last@cs.utlcluj.ro

Abstract: Reasoning is central to the idea of the Semantic Web and ontologies, however, the fundamental principles of reasoning – soundness and completeness – do not match the reality of the (Semantic) Web that is ruled by contradicting and incomplete data and claims. Furthermore, logical reasoning is strong for rather small numbers of axioms and facts, while the Web is growing at an impressive speed and hence offering tremendous amounts of data. ‘Reasearch’ is a novel idea of combining reasoning with information retrieval methods (search) in order to respond to the requirements and constraints implied by reasoning on the Semantic Web. The Large Knowledge Collider is a modular reasoning platform that allows for reasearching with Web-scale data, and instrumenting and monitoring the platform is essential for verifying and assuring high performance, adaptability and well-functioning. These aspects are vital for experiments running on the platform.

Keywords: Reasoning, Semantic Web, Instrumentation and Monitoring.

1. Introduction

The Large Knowledge Collider, in short LarKC, is an experimentation platform for large-scale reasoning in the Web context. The platform is developed in response to the eminent mismatch between computation on the Web and logical reasoning [1]. Although reasoning for the Semantic Web is claimed to bring value and sense to the tremendous amount of information on the Web, the reality looks much less promising. Indeed, there is a significant disparity between the underlying principles of the Web and the ones of logical reasoning. Logical frameworks provide answers of truth to rather small numbers of axioms and small numbers of facts; e.g., Description Logic [2] scales well for concept definitions, but poorly for large instance sets, Logic Programming [3] engines can deal well with larger sets of axioms and instances, but can only draw very simple logical conclusions. The Web on the other hand naturally results in potentially infinite numbers of axioms and huge numbers of facts – considering the entirety of human knowledge and the still growing number of Web sites. A much more critical aspect, moreover, with regards to reasoning on the Web, is the requirement for completeness and correctness of inference rules that is assumed in logics. These characteristics can never hold on the Web, as it is open and naturally unbound. Collecting all relevant information from the Web is impossible, nor desirable. Furthermore, information on the Web is unreliable from the very beginning, as it is uncontrolled and

represents various view points on the same topic.¹ Reasoning on top of incomplete and contradicting Web content seems hence unfeasible, and even meaningless.

LarKC intends to counteract this discrepancy in the basic assumptions between reasoning and the (Semantic) Web by fusing reasoning and search into a paradigm that is referred to as *reasearch* [1]. The only way to allow for reasoning at Web scale is to interweave the reasoning process with retrieval and abstraction in order to select small subsets of the available information that still permit an acceptable result to the problem at hand. To this end, LarKC exploits the space between search and reasoning that is delimited by the curves of precision/recall as of information retrieval, and soundness/completeness as of reasoning (Figure 1); i.e., retrieval and reasoning become two sides of the same coin.

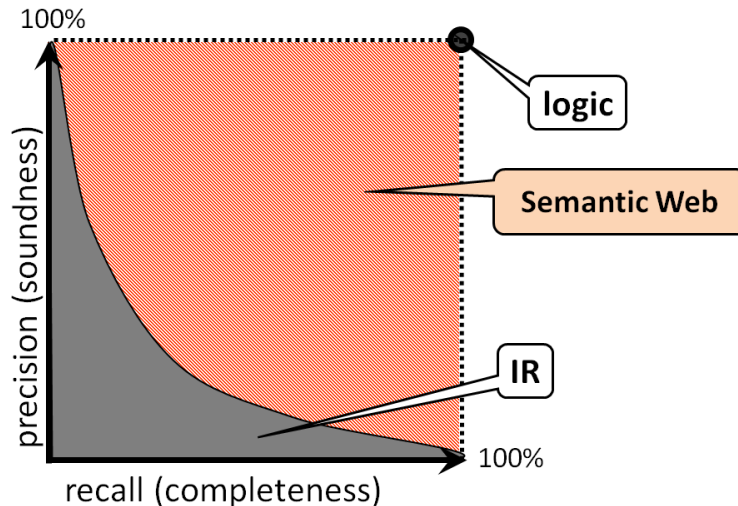


Figure 1: Positioning LarKC – reasearch between IR and logics

The goal of LarKC is thus to provide a platform for massive distributed incomplete reasoning that removes the scalability barriers of traditional reasoning systems for the Semantic Web. This is achieved through the enhancement of logic-based reasoning with information retrieval or machine learning methods that trade-off the 100%-requirement in soundness and completeness of logics in exchange for lowered costs in computation. More concretely, LarKC yields a pluggable infrastructure that ensures that computational components of diverse fields can be coherently integrated (retrieval and selection methods from IR, cognitive science or statistics; abstraction from machine learning and knowledge modelling; or decision methods from economics or decision theory), in order to coordinate large scale inference over distributed and heterogeneous information and resources.

2. Objectives

In this paper we present instrumentation and monitoring methods that are commonly used in Web-based querying and inference systems, and show how this methods yield an instrumentation and monitoring platform that complements the LarKC infrastructure. The pluggable nature of LarKC allows for flexible composition of research processes (so-called workflows of special-purpose plug-ins) and the dynamic acquisition of compute resources both on high-performance computing clusters or Web servers, and via “computing at home” as for example known from SETI@home. The two aspects of pluggability and distribution are essential characteristics of LarKC’s conceptual architecture.

¹ Investigating possibilities to model and exploit the diversity of knowledge on the Web is the main topic of the recently established research project RENDER (render-project.eu).

While the LarKC platform provides the coordination facilities and the main tools for building workflows, and hence for addressing complex and large scale research problems, the actual computation is done in the plug-ins. From the perspective of LarKC, plug-ins are simple compute containers that wrap and manage the methods exposed to the reasoning infrastructure. The role of LarKC is then to seamlessly integrate the plug-ins of various purposes from very diverse fields, as they were listed above. To do so, the infrastructure relies on a plug-in annotation ontology that allows for describing the functionality and interfaces of plug-ins, and that provides means to specify how plug-ins can be run in different execution environments including the parallelization of execution; e.g., local execution or remote execution on servers or parallelization on high-performance clusters. On the basis of their semantic descriptions, plug-ins can be easily integrated with the workflow specifications. These determine how plug-ins are plumbed together and how the output of a reasoning process is consumed. The plug-in and workflow descriptions, as well as the processed information are all provided in RDF, and the default sink for LarKC processes is a SPARQL endpoint [4] – other endpoints such as anytime or publish/subscribe are also possible. In summary, the LarKC platform offers the technicalities to describe plug-ins and workflows, and the components to construct integrated search and reasoning processes that are run under the control of an executor; the LarKC platform instantiates one executor per workflow, which manages the entire life-cycle of its workflow (Figure 2).

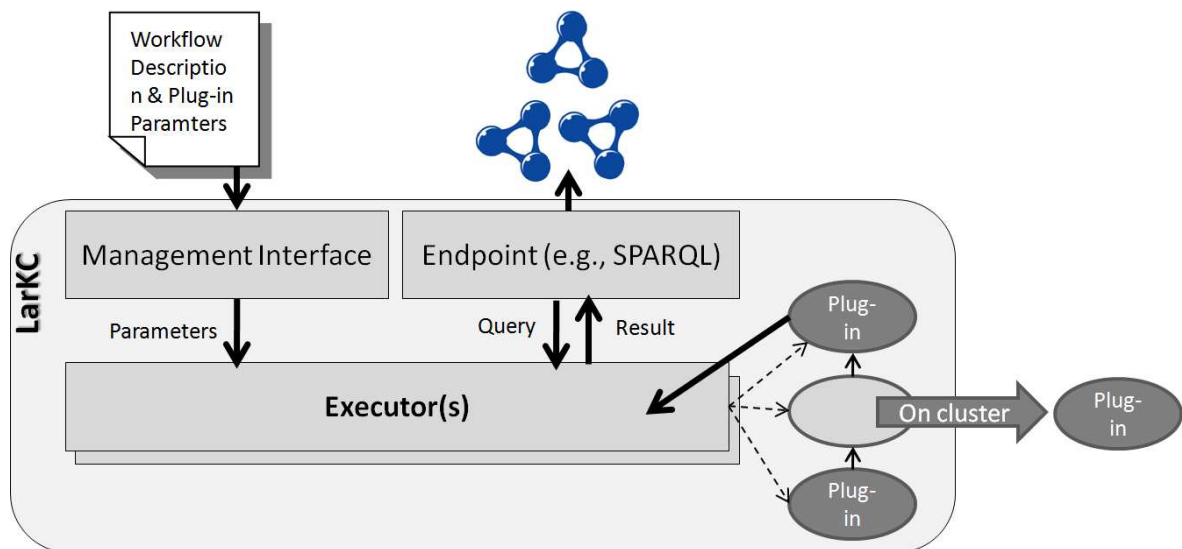


Figure 2: LarKC platform architecture

The proposed instrumentation and monitoring platform, which is subject to this paper, provides then a consistent set of interfaces and data representations to monitor the performance of plug-ins, workflows and the platform (e.g. speed of triple retrieval, cache efficiency of the data-layer, data transport overhead) in real-time with visualization to support rapid experimental prototyping and with high-performance, low overhead logging to support detailed scientific evaluation. For example, plug-in for probabilistic inference may want to record the degree of internal parallelism it achieves for SAT solving - the experimental facility will allow for this data to be captured and stored for analysis without significant programming overhead on the plug-in writer or significant performance overhead. Such meta-analysis enables experiments in Web scale reasoning for which plug-ins or workflows learn to improve their own performance by analyzing multiple possible configurations.

In the remainder of the paper, we consider in more detail the instrumentation and monitoring solution of LarKC and discuss how it supports the objective of LarKC in providing large-scale reasoning over incomplete and inconsistent data on the Web.

3. Models and Metrics for Instrumentation and Monitoring

The growing complexity of software and hardware has increased the importance of instrumentation and monitoring in modern day programming practices. In a nutshell, instrumentation provides code that enables measurements of various system and program related metrics that are of interest to the user. Monitoring represents the process of performing the measurements and making the system users aware of the data. Approaches in instrumentation can be generic (model-based) or specific (related to different tasks such as debugging, tracing, or logging). Model-based approaches are independent of programming languages and the instrumentation process is based on generic instrumentation patterns (e.g., RPC-style request/response interaction patterns, synchronous and asynchronous messaging patterns).

The methodology followed in LarKC for the development of instrumentation and monitoring for large scale reasoning systems includes the survey of existing approaches, the identification of relevant metrics for LarKC users and developers, the design of a scalable distributed architecture that specifies components and their interfaces, the implementation of this architecture, and finally the implementation and testing of the instrumentation and monitoring solution in real-life settings; e.g., LarKC showcases on urban computing or cancer research.

3.1 – Types of Metrics

The nature and use of the LarKC platform and its components requires the coverage of several categories of metrics:

- Performance metrics refer to execution time (the time during which actual work has been carried out) or throughput (estimated rate of generated results).
- Resource-related metrics address the usage of resources and the infrastructure where the plug-ins are executed, including memory consumption, CPU usage, number of threads.
- Invocations and instance metrics represent quantity metrics; i.e., information about invocation counts of methods, plug-ins, workflows or instances of an entity. Also counted are the number of instantiations of a given plug-in, the number of queries sent to the platform in a certain amount of time, the number of accesses to the data layer, how many successful executions have been run, how many failures and others.
- Data metrics capture the size of the data that can be input or output to a task/method or transmitted between plug-ins.
- Events-related metrics are properties associated to special exceptions and errors for recording a successful execution, a failure, or a thrown exception.

The above list of metrics is based on previous work done in the field. Most approaches comprise relevant metrics for the domain of instrumentation. For example, [5] proposes resource usage characteristics, such as CPU usage, memory usage, disk usage that were used to make experiments with HP Neoview database systems, [6] uses resource metrics to measure the performance of multi-core systems, while [7] use resource metrics and threads related information to estimate the performance of instrumentation techniques for threaded programs.

3.2 Modelling the metrics

The chosen metrics that are relevant for LarKC developers and users are grouped in several categories, based on different classification criteria. Different metrics belonging to one category according to one criterion can be part of another category according to the other criteria. The first criterion we look at is the specificity; it refers to how general or

specific metric is with regards to the system that is being instrumented. Based on the specificity the following metrics are distinguished:

- Generic metrics are common to many Java applications and include:
 - Method-related metrics such as WallClockTime (elapsed time between method entry and method exit), CPU-related metrics, and number of threads created.
 - JVM-related metrics capture information that can be measured by instrumentation in direct mode per JVM by sampling at a given interval. Some of the metrics are related to the number of classes loaded or compilation time.
 - System metrics are measured by sampling, every number of seconds, the system or node information. Examples of metrics in this category are related to system load, free or used memory, and swap space.
 - Instrumented application metrics are measured at application start and offer data about the operating system, IP address, application name, virtual machine version.
- LarKC specific metrics are special to particular plug-ins, workflows or the platform

Complexity is the other criterion that is considered for formalizing the metrics model. It refers to how metrics measurements are obtained and it splits metrics into two classes: atomic and compound metrics.

- Atomic or simple metrics are directly measured by the instrumentation code and are grouped as follows:
 - Query-related metrics comprise information about the query such as the number of RDF triples, size of query, namespaces mentioned in the query, number of variables, time elapsed since the query was provided to the platform.
 - Workflow metrics contain information about the duration of the workflow, about the plug-ins it contains, about the threads started.
 - Plug-in metrics capture information about the size of data transmitted between plug-ins, number of accesses to the data layer, exceptions thrown by plug-ins.
 - Platform metrics gather data related to execution time of the platform, average CPU load during a given time interval, average memory usage.
- Compound metrics are obtained by combining atomic metrics or by applying different mathematical operators to the atomic metrics (in most of the cases a summation or division is applied). Some examples, among many, of possible combinations are the number of queries that were received by the LarKC platform in a given time interval, the number of workflows that were run in a given time interval, or the average dimension of input data for a plug-in.

4. Instrumentation and Modelling Components and Technologies

The overall architecture of LarKC instrumentation and monitoring for large scale reasoning systems is depicted in Figure 3. The components that are part of the architecture implemented use various technologies (e.g. aspect oriented programming, agent technologies, semantic technologies) and are aware of the metrics and models introduced in the previous section.

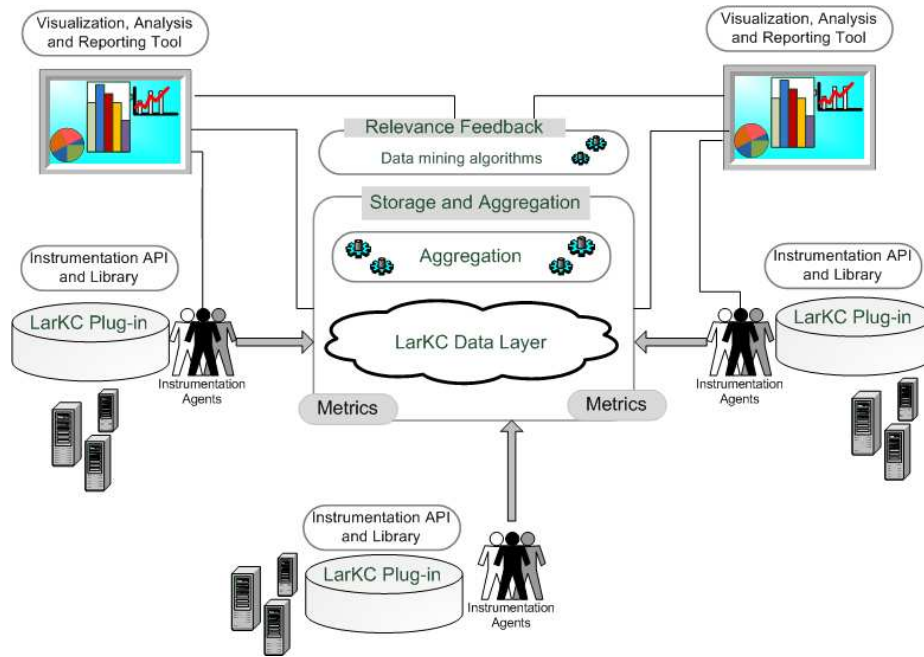


Figure 3: Instrumentation and monitoring - schematic view

The LarKC instrumentation and monitoring solution for large scale reasoning systems includes the following components that are discussed in more details in the remainder of this section: instrumentation / profiling agents, instrumentation API and library, storage and aggregation, visualization and relevance feedback.

4.1 – Instrumentation Agents

Software artifacts that are deployed close to the resources being monitored are called in our approach *instrumentation agents* or *profiling agents*. Their main role is to collect and buffer the measurement data that is supplied by the instrumentation code. The agents are asynchronous or non-blocking, meaning they permit the monitored component to continue the work without interruption. Furthermore, the profile agents are easily customizable, extendable and adaptable to consider new metrics or updated definitions of existing metrics. The agents send data to the storage component, the central point where the data is kept. Existing solutions based on agents include for example Usermon, Hyper HQ, and Zabbix,² and have been considered as starting point for the instrumentation agents. In our approach the agents and the instrumentation code are implemented using Aspect oriented programming (AspectJ³).

4.2 – Storage and Aggregation

The storage and aggregation server is a core component responsible for the storage and aggregation of the monitored data received from the profiling agents. Furthermore, the server provides the means to access, to retrieve and to query the measurement data.

Monitoring data will be published in the LarKC data layer as instances of the identified metrics. On top of the raw data that is send by profiling agents an aggregation processing component offers users summaries and complex views of recorded metrics' information. The data stored by the storing component and processed by the aggregation component will be exposed through SPARQL endpoints.

² <http://code.google.com/p/usemon>; <http://www.hyperic.com>; <http://www.zabbix.com>

³ <http://www.eclipse.org/aspectj/>

4.3 – Instrumentation API and Library

The instrumentation mechanism should be available for all LarKC users. That is why we provide an instrumentation API and library allowing LarKC plug-in developers to instrument the plug-in behavior. The API is planned to contain a minimal set of methods to allow programmatic specification of the instrumentation and will be accompanied by a library of processing code. Furthermore an annotation mechanism is provided to the developers allowing them to specify which methods should be instrumented, which metrics should be measured, how often, etc.

4.4 – Visualization

Nowadays we can distinguish between different areas of visualization [8]:

- Information visualization: is usually applied to the visual representation of large-scale collections of non-numerical information, such as bibliographic databases, files, lines of code in software [9], ontology models.
- Scientific visualization: uses visualization of phenomena in architecture, medicine, biology, etc.
- Data visualization: includes representation of data, which has been abstracted in some schematic form, including attributes or variables for the units of information.

The role of the visualization component is to provide an integrated view of the stored and aggregated data. It will enable an easy interpretation and analysis of the data. Using this tool LarKC users and developers are able to visualize real time or historical data related to queries, workflows or plug-ins. Furthermore they can analyze reports based on aggregated metrics. The data being visualized is fetched from the storage and aggregation component. Another functionality of the visualization component is to allow users the possibility to input some query or work-flow configurations that will be send to the relevance feedback mechanism which, in its turn, will communicate to the visualization tool some data (in the form of metrics, or plug-in names, or workflow parameters).

4.5 – Relevance Feedback

The relevance feedback component uses data mining techniques and performs machine learning on top of the instrumented data. Similar approaches have been used to predict workload parameters for multicore systems [5], distributed databases [8], networking [10].

The purpose of the relevance feedback mechanism is to analyze the feature space (features are metrics relevant for instrumentation and monitoring of large reasoning systems) and perform operations like: grouping similar data (using clustering algorithms like Expectation Maximization clustering or k-means clustering) or predicting some metric values based on other metrics (using linear regression models). LarKC is an experimental, modular platform for large-scale reasoning in which various components (e.g. reasoning, selection, etc.) can be plugged in and tested. The relevance feedback mechanism can be used to monitor how does a particular plug-in behave based on different inputs (queries about different concepts, different types of queries, sizes of datasets mentioned in the query or related to the query), in different circumstances (number of concurrent plug-ins running on the system, load of the system responsible for executing the plug-in) and what do they output (e.g. number of results). The behavior would be measured in terms of time or system resources used by plug-in and success of the operation.

5. Business Benefits

Instrumentation and monitoring of distributed platforms and services is a topic of high interest that could provide real business benefits if available. Providing instrumentation and monitoring tools to developers brings a significant reduction of the development time, development costs and results in the end in better applications and services being made available on the market in a shorter time. Furthermore, for platforms and services that are in production, monitoring tools provide easy and efficient observation of these applications and services being resulting in lower costs for maintenance and management. For large scale reasoning platform in particular, having instrumentation and monitoring is even more critical in business settings since for these types of platforms, performance, resource consumption as well as other metrics need to be closely monitored and consider in order for maintaining a competitive advantages over other businesses.

6. Conclusions

A formal perspective upon monitoring and instrumentation for large scale reasoning has been introduced in this paper. The LarKC instrumentation and monitoring solutions provide easy means to instrument applications, using predefined as well as customer defined metrics, to collect, aggregate, query and visualize measurements data represented in RDF in a very flexible way. Such instrumentation is essential in optimizing and monitoring LarKC workflows, i.e., data management processes for the extraction, maintenance and reasoning with linked data.

The LarKC platform itself is an open-source software project, and is intended to be freely available and supported beyond project termination. From a business perspective, the argument stands that the platform shall provide a core data management and reasoning infrastructure on top of which various services and plug-ins are implemented that generate business value, and hence financial profit. The purpose of LarKC is then to simplify the integration and shorten the time to market for novel data management and reasoning software that is made available via LarKC in form of plug-ins.

The work presented in this paper is supported by the ICT FP7 project LarKC. The authors express gratitude to their colleagues of the project, in particular those involved in the platform design and implementation: Frank van Harmelen, Michael Witbrock, Spyros Kotoulas, Matthias Assel, Alexey Cheptsov, Blaz Fortuna and Luka Bradesko.

References

- [1] D. Fensel, F. Van Harmelen: Unifying Reasoning and Search to Web Scale. *IEEE Internet Computing* 11(2), 2007.
- [2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider: *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press, 2003.
- [3] J. W. Lloyd: *Foundations of Logic Programming*. Springer-Verlag, 1987.
- [4] E. Prud'hommeaux, A. Seaborne: *SPARQL Query Language for RDF*. W3C Recommendation, 2008.
- [5] A. Ganapathi, H. Kuno, U. Dayal, J.L. Wiener, A. Fox, M. Jordan, D. Patterson: Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. *IEEE Int'l Conf. on Data Engineering*, 2009.
- [6] A. Ganapathi, K. Datta, A. Fox, D. Patterson: A Case for Machine Learning to Optimize Multicore Performance. *1st USENIX Workshop on Hot Topics in Parallelism*, 2009.
- [7] Z. Xu, B.P. Miller, O. Naim: Dynamic Instrumentation of Threaded Applications. *7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1999.
- [8] M. Friendly, D.J. Denis: Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization. Online at <http://datavis.ca/milestones/> (Accessed: February 2011).
- [9] S.G. Eick: Graphically Displaying Text. *Journal of Computational and Graphical Statistics* 3(2), 1994.
- [10] M. Couceiro, P. Romano, L. Rodrigues: A Machine Learning Approach to Performance Prediction of Total Order Broadcast Protocols. *IEEE Int'l Conf. on Self-Adaptive and Self-Organizing Systems*, 2010.