Executing Evaluations over Semantic Technologies using the SEALS Platform

Miguel Esteban-Gutiérrez, Raúl García-Castro, Asunción Gómez-Pérez

Ontology Engineering Group, Departamento de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid, Spain
{mesteban,rgarcia,asun}@fi.upm.es

Abstract. The SEALS European project aims to develop an infrastructure for the evaluation of semantic technologies. This paper presents in detail the approach followed to automate the execution of these evaluations in the infrastructure. To materialize this approach, we have defined the entities managed by the infrastructure and their life cycle, the process followed to execute evaluations, the management of the computing resources that form the execution infrastructure, and how tools can be integrated with the infrastructure.

1 Introduction

The SEALS European project is developing an infrastructure for the evaluation of semantic technologies, named the SEALS Platform, that will offer independent computational and data resources for the evaluation of these technologies.

With the SEALS Platform users will define and execute evaluations on their own and will support the organization and execution of evaluation campaigns, i.e., worldwide activities in which a set of tools is evaluated according to a certain evaluation specification and using common test data.

One of the challenges in the development of this platform is to cope with the different heterogeneous semantic technologies and with the different evaluations that could be performed over them. On the one hand, this requires reconciling heterogeneity in the technical level, where we need to execute evaluations by uniformly accessing semantic technologies with different hardware and software requirements. On the other hand, in the information level we need to achieve a common understanding of all entities that participate in the evaluation process.

This paper presents how the execution of evaluations over semantic technologies is performed in the SEALS Platform and the mechanisms defined and developed to achieve it. These mechanisms involve the definition of the materials needed during the evaluation, together with the format in which they have to be provided so they can be used within the platform, to the provision of an automated way of managing the tools to be evaluated, the evaluations to be performed over the tools, and the computing infrastructure where the evaluations will be carried out.

This paper is structured as follows. Section 2 introduces our approach to automatically manage evaluations over semantic technologies. Section 3 gives

Proceedings of the International Workshop on Evaluation of Semantic Technologies (IWEST 2010). Shanghai, China. November 8, 2010.

an overview of the architecture of the SEALS Platform and section 4 presents the entities managed in the platform and their life cycles. Section 5 describes the process followed for processing an evaluation using the platform. Section 6 explains the execution infrastructure used in the SEALS Platform, that is, how the computing resources used in the platform are managed and what is the life cycle of such resources. Section 7 describes how to integrate tools with the SEALS Platform so they can be used in evaluations. Finally, section 8 provides some conclusions of this work.

2 The SEALS Approach to Software Evaluation

As illustrated in Figure 1, in any *evaluation* a given set of *tools* are exercised, following the workflow defined by a given *evaluation plan* and using determined *test data*. As an outcome of this process, a set of *evaluation results* is produced.

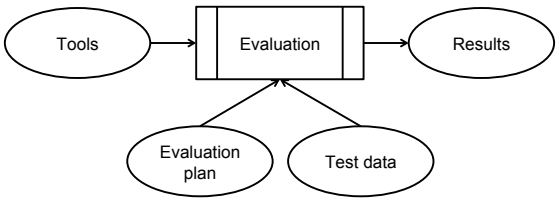

Fig. 1. Main entities in a software evaluation scenario.

This idea of software evaluation is largely inspired by the notion of evaluation module as defined by the ISO/IEC 14598 standard on software product evaluation [1].

The general process of executing a software evaluation comprises:

1. To prepare all the evaluation materials, having them in the appropriate format and accessible to be used in the evaluation.
2. To prepare and configure the evaluation infrastructure, that is, to set up the computing resources where the tools under examination will be executed during the evaluation process. In this context, the infrastructure refers to both hardware and software dimensions of a computing resource.
3. To deploy the tools to be evaluated in the evaluation infrastructure. This requires installing the tool in a suitable computing resource and configuring the tool appropriately, paying special attention to the integration of the tool with required third party applications in the evaluation infrastructure.
4. To define the evaluation plan, identifying the tasks that have to be carried out and the order in which each of them is to be performed. For each task it is necessary to define the set of input test data as well as the expected results. The plan must also identify the control conditions that allow progressing between the tasks, which can be summarized as the set of the success/failure

conditions for each of the tasks and also the pre and post conditions for the execution of a particular task.
5. To execute the evaluation plan to obtain the evaluation results.

Clearly, all these steps could be performed manually. However, the manual evaluation of software does not scale when one tool has to be evaluated several times over time, when different tools have to be evaluated following the same evaluation plan, or when different tools have to be evaluated following different evaluation plans, which is our goal.

The SEALS Platform aims to automate most of the software evaluation process. To this end we need to have:

- All the materials needed in the evaluations represented in a machine-processable format and described with metadata to allow their discovery and access.
- Automated mechanisms to prepare and configure the infrastructure required for executing tools with different hardware and software requirements.
- Automated mechanisms to install tools in the evaluation infrastructure and to configure them.
- Formal and explicit evaluation plans provided as machine-processable specifications, also known as *evaluation descriptions*.
- Means to automatically execute such *evaluation* plan *descriptions*, which includes the interaction with the tools.

3 Overview of the SEALS Platform

The architecture of the SEALS Platform comprises a number of components, shown in Figure 2, each of which are described below.

- **SEALS Portal.** The SEALS Portal provides a web user interface for interacting with the SEALS Platform. Thus, the portal will be used by the users for the management of the entities in the SEALS Platform , as well as for requesting the execution of evaluations. The portal will leverage the SEALS Service Manager for carrying out the users' requests.
- **SEALS Service Manager.** The SEALS Service Manager is the core module of the platform and is responsible for coordinating the other platform components and for maintaining consistency within the platform. This component exposes a series of services that provide programmatic interfaces for the SEALS Platform. Thus, apart from the SEALS Portal, the services offered may be also used by third party software agents.
- **SEALS Repositories.** These repositories manage the entities used in the platform (i.e., test data, tools, evaluation descriptions, and results).
- **Runtime Evaluation Service.** The Runtime Evaluation Service is used to automatically evaluate a certain tool according to a particular evaluation description and using some specific test data.

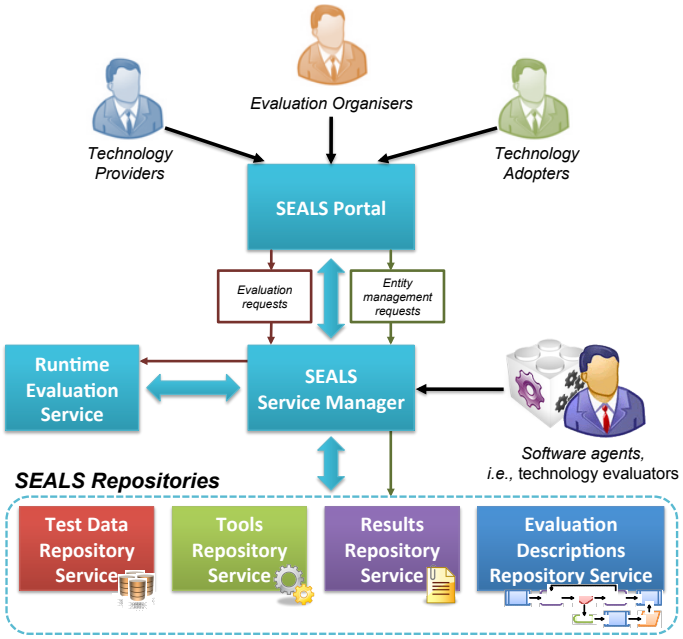

Fig. 2. Architecture of the SEALS Platform.

4 The SEALS Evaluation Entities

The high-level classification of software evaluation entities presented in section 2 can be further refined as needed. For example, in the context of SEALS, tools are classified into different types of semantic technologies according to their functional scope, namely, *ontology engineering tools*, *ontology storage systems*, *ontology matching tools*, etc.

Similarly, it is also possible to distinguish different types of test data: *persistent test data* (those whose contents are stored in and physically managed by the evaluation platform), *external test data* (those whose contents reside outside the evaluation platform and whose life cycle is not controlled by it), and *synthetic test data generators* (pieces of software that can generate synthetic test data on-demand according to some determined configuration).

In accordance with the approach followed in the IEEE 1061 standard for a software quality metrics methodology [2], evaluation results are classified according to their provenance, differentiating *raw results* (those evaluation results directly generated by tools) from *interpreted results* (those generated from other evaluation results).

Besides, our entities include not only the results obtained in the evaluation but also any contextual information related to such evaluation, a need also acknowledged by other authors [3]. To this end, we also represent the information required for automating the execution of an evaluation description in the

platform that, with the other entities presented, allows obtaining traceable and reproducible evaluation results.

Finally, another type of entities are *evaluation campaigns*, which represent the information needed to support the organization and running of campaigns for the evaluation of different (types of) participating tools. An evaluation campaign contains one or more *evaluation scenarios*, which include the evaluation description and test data to be used in the evaluation and the tools to evaluate.

Each of the abovementioned entities is composed of two different elements: the *data* that define the entity itself and the, for instance, a set of ontologies serialized in RDF/XML in the case of an *persistent test data set*; and *description* of the entity, that is, the set of metadata that characterizes the entity (both generally and specifically) and enables the provision of the discovery mechanisms required for entity integration, consumption, and administration by the evaluation platform, i.e., in the previous example, the metadata would be the information about the purpose of the *persistent test data set*, which evaluations may use it, and who may access to the test data set.

Different entities have different life cycles in the evaluation platform. Next, we describe the life cycles of the most relevant entities.

4.1 Life Cycle of Artifacts and Artifact Versions

Tools, test data, and evaluation descriptions are defined in the platform as *artifacts*, which are collections of *artifact versions*; for example, a particular tool can have a number of different tool versions that evolve over time.

Figure 3 shows state diagrams for artifacts and artifact versions, including the possible states, the operations that alter the state, and the operations that retrieve the entity information (data and metadata) in dotted arrows. It can be observed that, once registered in the platform, artifacts can always be retrieved and have a single state until they are deregistered. On the other hand, artifact versions have two states, published and unpublished; in the former state artifact versions can only be retrieved, and in the latter they can only be updated. Hence, evaluations can only be performed using fixed (i.e., published) artifact versions.

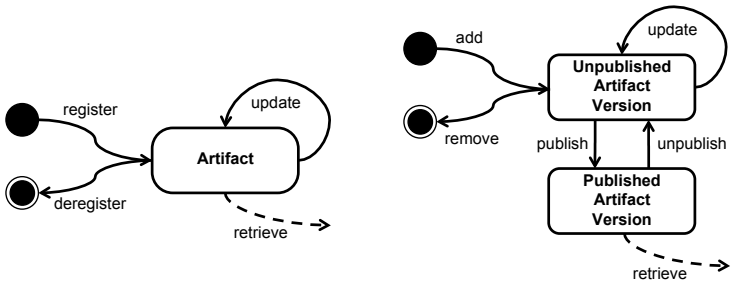

Fig. 3. Life cycle of artifacts (left) and artifact versions (right).

Evaluation results (raw results and interpretations) are defined as artifacts with no version information. Besides, once registered they cannot be updated.

4.2 Life Cycle of Execution Requests

Evaluation descriptions are processed by the evaluation platform through *execution requests*. An execution request encapsulates the execution needs that a particular user has at some point in time, i.e., the evaluation description to be executed, the tools to be evaluated, the test data to be used, etc.

During its life cycle, an execution request transits among eight different states, as shown in Figure 4. The starting state of an execution request is that of *"pending"*, which takes place whenever a new execution request is created.

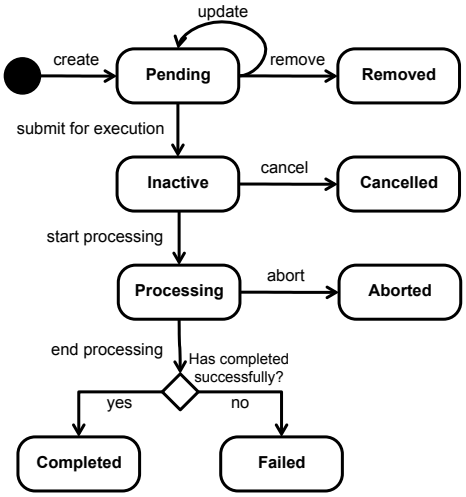

Fig. 4. Life cycle of an execution request.

At this point, the execution request can be updated, removed, or submitted for execution. Whereas the first operation does not change the state of the execution request, the other two do change it: on the one hand, when the execution request is removed, the state transits to the *"removed"* state, a state in which no further operations are possible[1]; on the other hand, when the execution request is submitted to execution, the state transits to the *"inactive"* state. Beyond this point, the execution request shall not be further modified.

While the execution request is inactive, two possible courses of action can take place: it can be cancelled or it can start being processed. In the first case, the state transits to the *"cancelled"* state, a state in which, again, no further

[1] That is, the state of the execution request will not be changed beyond this point.

operations are possible. The latter case takes place once the execution requirements of the execution request are fully satisfied and, then, the state transits to the *"processing"* state.

Once the execution request is being processed, three possible outcomes may occur: (1) The execution request may be completed successfully, and thus the state transits to the *"completed"* state. (2) Some failure might prevent completing the execution of the execution request, causing the state to transit to the *"failed"* state. (3) It is also possible that the processing of the evaluation request is aborted (e.g., due to an abnormal duration time), thus forcing the state to transit to *"aborted"*. Regardless of the course of action, no further operations over the execution request will be carried out.

As can be seen, execution requests are not disposed by the evaluation platform. On the contrary, regardless of the execution request's internal state, its information is available to the user at any time, providing a complete and historical view of the evaluation activities over time.

5 Processing Evaluations

Processing an evaluation *execution request* consists in executing the evaluation description associated with that request. The process required for accomplishing such task is carried out in four stages, as shown in Figure 5, namely, *evaluation description analysis*, *execution environment preparation*, *evaluation description execution*, and *execution environment clean-up*. The following subsections will cover each of the stages of this process.

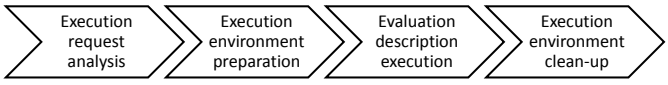

Fig. 5. Stages for the processing of an execution request.

5.1 Execution Request Analysis

In this very first stage, the Runtime Evaluation Service analyses the execution request in order to guarantee that it can be processed, and prepares all the information that is required for driving the rest of the evaluation process.

Among other things, the analysis includes: (1) Checking the evaluation description, i.e., validating the syntax of the evaluation description and checking that the work-flow described is well-defined; (2) Checking if the execution request arguments satisfy the evaluation description contract, i.e., verifying the availability and type of the specified entities. If any of these verifications fails (syntax, semantics, or resources), the stage will fail and thus the processing of the execution request will completely fail. Otherwise, the stage will successfully complete and trigger the next stage.

5.2 Execution Environment Preparation

Once the Runtime Evaluation Service has checked that the execution request may be safely executed, it is time to prepare the execution environment, that is, to prepare the set of *computing resources* where the tools to be exercised during the execution of the evaluation description will be physically run.

In this context, a *computing resource* is any network accessible computing appliance that shall be used for the on demand execution of tools, and exposes a series of mechanisms for its remote administration and usage. Examples of this are desktop PCs, workstations, servers, and even virtual machines running atop the previously mentioned appliances.

Since each tool may have its own computing requirements (i.e., a determined operating system or a particular third party application) and the computing resources available will be limited, computing resources have to be reused.

In order to enable the usage of the computing resources the SEALS Service Manager provides the means for tracking the availability of resources, as well as for its lease and release, which requires to provide the means for describing the characteristics of these resources so that it is easy to choose those which better fit for the execution of a given tool.

In order to enable the reuse of these shared resources, the Runtime Evaluation Service will be in charge of preparing the computing resources according to the requirements of the tools under evaluation, and this will be carried out in two steps. First, the Runtime Evaluation Service will request from the SEALS Service Manager the computing resources that will be needed for executing the tools involved in the evaluation description (see section 6). Then, once the computing resources have been acquired, the Runtime Evaluation Service will have to deploy in them the tools to be used during the execution of the evaluation description, as well as to deploy any third party application required by the tools.

5.3 Evaluation Description Execution

In this stage the Runtime Evaluation Service enacts the workflow defined in the evaluation description following the defined flow of control and executing the activities specified within the workflow.

The execution of these activities is composed of one or more of the following steps, depending on the specific activity and the current state of execution:

1. The first step is to stage-in all the data to be used in the activity, in other words, making the data involved available in the computing resource where the activity will be executed.
2. Once the data is available it is time to execute the particular activity. The activity can imply invoking a tool's functionality or the interpretation of raw results by means of specific software artifacts, the *interpreters*.
3. Regardless of the specific activity executed, the next step consists in storing the results obtained in the *Results Repository*. These results will be raw results if the activity executed was the invocation of a tool's functionality, and interpreted results otherwise.

4. Finally, any data that is not going to be further used in the computing resource should be deleted and thus the storage space freed. This final step will occur even if any previous step has failed to complete.

If any of these steps fails, whichever the cause, the evaluation description execution stage will fail, and thus the processing of the execution request will fail to complete. Otherwise, the stage will complete successfully and trigger the *execution environment clean-up* stage.

5.4 Execution Environment Clean-up

In this last stage, the Runtime Evaluation Service will ensure that the shared computing resources can be reused after processing the execution request. This stage is carried out in two steps.

The first step in the clean-up consists in removing the tools that have been previously deployed in each of the computing resources acquired with the objective of leaving each computing resource in the same state as it was before deploying the tools.

Finally, after ensuring that the computing resources used are in the same state as they were when acquired, the Runtime Evaluation Service will release them, acknowledging this to the SEALS Service Manager, who will then be able to lease again these computing resources for further reuse.

6 Evaluation Infrastructure

The purpose of the SEALS Platform is the automated evaluation of semantic tools, being the *Runtime Evaluation Service* the component in charge of carrying out the process. Thus, this service is responsible for executing the tools that are under evaluation. SEALS deliverable D9.1 [4] defines the way in which this component will interact with them.

In order to be executed, a tool may require using determined third party applications or tools, from now on referred to as *modules*. To this respect, the Runtime Evaluation Service identifies two types of runtime dependencies: *internal dependencies* and *external dependencies*. The former type refers to those modules provided in the tool's package, e.g., a given third party library. The latter type refers to those modules not provided in the tool's package and thus the platform must provide in the execution environment, e.g., a DBMS.

This division is aimed at solving the "deployment" issue. This way, technology providers can include in the tool's package those modules that can be deployed without user intervention, and rely on the platform for providing those modules whose deployment is much more complex, or requires user intervention.

The SEALS Platform will have to publish which modules are provided in the execution environment, so that technology providers are informed about what they can use and act in consequence requesting the addition of new modules or implementing the means for deploying the module dynamically.

Regardless of the type of dependency, the configuration of the runtime dependencies of a tool (tool wrapper to platform and vice versa) will be carried out via the package descriptor.

6.1 Management of Computing Resources

To execute different types of tools under variable circumstances, it is necessary to provide different *computing resources*, which shall cope with some sensible aspects related to the execution environment: operating system, computing power, memory size, storage size, and execution supporting modules. The SEALS Service Manager needs to manage these computing resources, leasing these resources as required by the Runtime Evaluation Service.

Our approach for managing computing resources has been to provide an array of virtual machines that are allocated to different physical computing resources. Each of these virtual machines will have a different set of characteristics and will be used exclusively for executing a single tool at a time.

The advantages of this approach are the following. First, thanks to virtualization the usage of the physical computing resources would be maximized, as any physical computing resource could execute any required virtual machine snapshot, thus enabling the scalability of the solution. Second, the usage of physical computing resources enables: speeding up tool execution; trustfully comparing performance metrics; coping with an increasing number of required features from technology providers, since a pool of specific virtual machine snapshots could be used. Third, an allocation policy is not required, as physical computing resource sharing is not allowed. And fourth, there is no single point of failure, as each virtual computing resource would be run on its own virtualized environment.

Nevertheless, this approach still has some drawbacks. First, performance is still an issue; the need for a virtual machine manager mediating between the execution environment and the physical computing resource is still a factor to take into account. Second, using the physical computing resources exclusively does not prevent resource underutilization completely. Finally, this additional mediation layer is still a risk from the architectural standpoint.

6.2 Life-cycle of Computing Resources

As presented above, the computing resources that will be provisioned by the platform consist of virtual machines running atop a collection of physical computing resources. Each virtual machine will run a particular snapshot of a given virtual machine image. These images define both the hardware and software characteristics of the computing resource, since they define the physical resources to be used when running the virtual machine (processors, memory, disk, etc.) as well as the operating system together with the preinstalled applications.

Next, we provide an overview of the life-cycle of computing resources:

– **Provisioning computing resources**. To provision computing resources, the SEALS Service Manager needs to deal with two different entities: physical

computing resources and virtual machine images. Thus, the SEALS Service Manager will maintain registries for both entities and will identify the virtual machine image that suits the requirements of the Runtime Evaluation Service and will allocate it to any of the available physical computing resources that fulfil the hardware requirements of the virtual machine image.
Once the virtual machine image has been allocated to a physical computing resource, the SEALS Service Manager will start a virtual machine using the image in the physical computing resource. To do so, the physical computing resource exposes a series of management capabilities that allow deploying a virtual machine in it and controlling its life-cycle.
Finally, after deploying the virtual machine, the SEALS Service Manager will hand over the control of it to the Runtime Evaluation Service.

- **Consuming computing resources** To consume computing resources, the Runtime Evaluation Service needs mechanisms for acquiring these computer resources from the SEALS Service Manager. To this end, the Runtime Evaluation Service shall specify the characteristics of the computing resources needed and a time window for using the requested computing resources.
  Once the SEALS Service Manager leases the matching computing resources, the Runtime Evaluation Service will use an entry point service for discovering the services deployed on the computing resource that shall enable the usage of the computing resource during the processing of an execution request.
- **Decommissioning computing resources** Finally, whenever the Runtime Evaluation Service is done with a computing resource, it will release it. After being acknowledged, the SEALS Service Manager will stop the associated virtual machine, and deallocate the virtual machine image from the physical computing resource.
  In case that the maximum wall-clock limit is reached and the Runtime Evaluation Service has not released the computing resource, the SEALS Service Manager will decommission the computing resource and acknowledge the decommission to the Runtime Evaluation Service.

7 Integrating Tools with the SEALS Platform

The *Execution Worker* is the component of the Runtime Evaluation Service in charge of executing the tools used when running evaluation descriptions. In order to be usable by the Execution Worker, tools must meet certain integration criteria regarding the capabilities that have to be exposed to the platform.

From a functional point of view, the Execution Worker provides a plug-in framework for allowing the dynamic usage of a priory unknown tools, which is based on the usage of an extensible interface hierarchy which defines the set of operations that the Execution Worker requires for using a certain type of tool.

Thus, in order for a tool to be usable by the Execution Worker it needs to provide an entry point which implements the interfaces required according to the particular nature of the tool. This entry point, hereafter referred to as *tool wrapper*, is thus in charge of linking the tool to the platform, or from the opposite point of view, it decouples the Execution Worker from the tool itself.

The tool wrapper provides two kinds of capabilities, namely the tool management and the tool invocation capabilities. On the one hand, the management capabilities include those mechanisms that allow the integration of the tool in the evaluation infrastructure as well as for controlling the life cycle of the tool itself. In particular, the management capabilities provide the means for deploying and undeploying a tool, and for starting and stopping the tool. On the other hand, the invocation capabilities provide the mechanisms for invoking the particular functionalities that have to be provided by each particular type of tool. For the time being, the tool wrapper can be implemented in two different ways: using a set of shell scripts, or using Java applications.

However, not any arbitrary implementation of the abovementioned capabilities might be used by the Execution Worker. In order for it to be consumed, it has to be provided in the form of a *tool package* bundle. The bundle consists of a ZIP file with a given directory structure that includes the binaries of the tool itself, the binaries of other applications required when running the tool, and a *package descriptor*, which instructs the Execution Worker about how the tool wrapper is implemented and which are its dependencies, so that the evaluation infrastructure can be properly set up.

8 Conclusions

This paper presents the approach followed in the SEALS Platform for the automated evaluation of different semantic technologies according to different evaluations. Such approach is based on the usage of machine-processable evaluation descriptions that define the evaluation plan without any ambiguity and identifies when the tools have to be executed, how they are executed, with which test data, and which results are to be obtained and stored. Also, by means of the metadata about all the entities involved in the evaluation process it is not just possible to discover the entities that may participate in an evaluation but also to validate the entities involved in a particular execution.

Acknowledgements

This work has been supported by the SEALS European project (FP7-238975).

References

1. ISO/IEC: ISO/IEC 14598-6: Software product evaluation - Part 6: Documentation of evaluation modules. (2001)
2. IEEE: IEEE 1061-1998. IEEE Standard for a Software Quality Metrics Methodology. (1998)
3. Kitchenham, B.A., Hughes, R.T., Linkman, S.G.: Modeling software measurement data. IEEE Trans. Softw. Eng. **27** (2001) 788–804
4. Miguel Esteban Gutiérrez: Design of the architecture and interfaces of the Runtime Evaluation Service. Technical report, SEALS Project (2009)