Chapter XIV
Benchmarking in the Semantic Web

Raúl García-Castro
Universidad Politécnica de Madrid, Spain

Asunción Gómez-Pérez
Universidad Politécnica de Madrid, Spain

Abstr Act

The Semantic Web technology needs to be thoroughly evaluated for providing objective results and obtaining massive improvement in its quality; thus, the transfer of this technology from research to industry will speed up. This chapter presents software benchmarking, a process that aims to improve the Semantic Web technology and to find the best practices. The chapter also describes a specific software benchmarking methodology and shows how this methodology has been used to benchmark the interoperability of ontology development tools, employing RDF(S) as the interchange language.

Introduct Ion

The Semantic Web technology has considerably improved since the 1990's, when the first tools were developed; although it has mainly been applied in research laboratories, in recent years companies have started to be interested in this technology and its application.

To transfer the Semantic Web technology from the academia, its current niche, to the industrial world it is necessary that this technology reaches a maturity level that enables it to comply with the quality requirements of the industry. Therefore, the Semantic Web technology needs to be thoroughly evaluated both for providing objective results and for attaining a massive improvement in its quality.

Until recently, the Semantic Web technology was seldom evaluated; now, however, this technology is widely used and numerous studies concerning its evaluation have appeared in the last few years. So now it seems quite necessary

that researchers increase the quality of their evaluations and improve the technology collectively by benchmarking it, employing for this a methodological process.

Evaluating and benchmarking this technology within the Semantic Web can be quite costly because most of the people involved do not know how to carry out these processes and also because no standard nor agreed methods to follow now exist. On the other hand, since it is quite difficult to reuse the results and put into practice the lessons learnt in previous activities, it is necessary to develop new methods and tools every time this technology has to be evaluated or benchmarked.

Software benchmarking is presented in this chapter as a continuous process whose aim is to improve software products, services, and processes by evaluating and comparing them with those considered the best. Although software evaluations are performed inside the benchmarking activities, benchmarking provides some benefits that cannot be obtained from evaluations, as for example, the continuous improvement of software, or the extraction of the best practices used to develop the software.

Within the Knowledge Web[a] European Network of Excellence a new methodology for benchmarking Semantic Web technology has been developed; this methodology is now being adopted in different benchmarking studies and applied to the different types of Semantic Web technologies (ontology development tools, ontology alignment tools, ontology-based annotation tools, and reasoners). The methodology focuses on the special interests of the industry and research fields and on their different needs. At the end of the chapter, we describe how we have followed this methodology during one of the activities performed to benchmark the interoperability of ontology development tools, employing RDF(S) as the interchange language.

eVAlu At Ion And benchm Ark Ing In the l Iter Ature

s oftware e valuation

Software evaluation plays an important role in different areas of Software Engineering, such as Software Measurement, Software Experimentation or Software Testing. In this section, we present a general view of these areas.

According to the ISO 14598 standard (ISO/IEC, 1999), software evaluation is *the systematic examination of to which extent an entity is capable of fulfilling specified requirements*; this standard considers software not just as a set of computer programs but also as a set of procedures, documentation and data.

Software evaluation can take place all along the software life cycle. It can be performed during the software development process by evaluating intermediate software products or when the development has finished.

Although evaluations are usually carried out inside the organisation that develops the software, other independent groups such as users or auditors can also make them. When independent third parties evaluate software, they are usually very effective, though their evaluations can become very expensive (Rakitin, 1997).

The goals of evaluating software vary since they depend on each specific case, but in general, they can be summarised (Basili et al., 1986; Park et al., 1996; Gediga et al., 2002) as follows:

- To **describe** the software in order to understand it and establish baselines for comparisons.
- To **assess** the software with respect to some quality requirements or criteria and determine the degree of quality required from the software product and its weaknesses.

- To **improve** the software by identifying opportunities and, thus, improving its quality. This improvement is measured by comparing the software with the baselines.
- To **compare** alternative software products or different versions of a same product.
- To **control** software quality by ensuring that it meets the required level of quality.
- To **predict** new trends in order to take decisions and establish new goals and plans for accomplishing them.

It is understood that the methods to follow to evaluate software vary from one author to another and from one Software Engineering area to another. However, from the methods proposed in the areas of a) Software Evaluation (ISO/IEC, 1999; Basili, 1985), b) Software Experimentation (Basili & Selby, 1991; Perry et al., 2000; Freimut et al., 2002), c) Software Measurement (Park et al., 1996; IEEE, 1998), and d) Software Testing (Abran et al., 2004) we can extract a common set of tasks that must be carried out in software evaluations. These tasks are the following:

1. To establish the evaluation requirements by setting its goals, the entities to evaluate, and their relevant attributes.
2. To define the evaluation by explaining the data to collect, the evaluation criteria to follow, and the mechanisms to collect data and implement these mechanisms.
3. To produce the evaluation plan.
4. To execute the evaluation and to collect data.
5. To analyse the collected data.

benchmarking in the literature

In the last decades, the word benchmarking has become relevant within the business management community. The most well-known definitions of the term are those by Camp (1989) and Spendolini (1992). Camp defines benchmarking as *the search for industry best practices that lead to superior performance*; on the other hand, Spendolini has expanded Camp's definition by adding that benchmarking is *a continuous, systematic process for evaluating the products, services, and work processes of organisations that are recognised as representing best practices for the purpose of organisational improvement*. In this context, best practices are good practices that have worked well elsewhere, are proven and have produced successful results (Wireman, 2003).

These definitions highlight the two main benchmarking characteristics:

- Continuous improvement.
- Search for best practices.

The Software Engineering community does not share a common benchmarking definition but several. Here we present some of the most representative:

- Both Kitchenham (1996) and Weiss (2002) define benchmarking as a software evaluation method suitable for system comparisons. Whereas for Kitchenham benchmarking is *the process of running a number of standard tests using a number of alternative tools/methods and assessing the relative performance of the tools in those tests*, for Weiss benchmarking is *a method of measuring performance against a standard, or a given set of standards*.
- Wohlin et al. (2002) have adopted the benchmarking definition from the business world, that is, they consider benchmarking as a continuous improvement process that strives to be the best of the best through the comparison of similar processes in different contexts.

software benchmarking

In this section, we have followed the notions that support continuous improvement and search for best practices within business management benchmarking; these notions have led us to consider software benchmarking as a continuous improvement process instead of as a punctual activity. Equally important for us are the concept of comparing software through evaluations and that of carrying out the benchmarking activity through a systematic procedure.

All these concepts permit us to define **software benchmarking** as a continuous process whose aim is to improve software products, services, and processes by systematically evaluating and comparing them with those considered to be the best.

This definition, however, does not specify the type of the entities considered in benchmarking (software products, services or processes), nor does it determine the software life-cycle phase when benchmarking is performed, and nor does it explain who is responsible for benchmarking. However, software benchmarking is usually performed on software products already developed and is executed by their developers.

The reason for benchmarking software products instead of just evaluating them is to gain those benefits that cannot be obtained from software evaluations. When we evaluate software we can observe its weaknesses and its compliance to quality requirements. If, on the other hand, several software products are involved in the evaluation, then we can have a comparative analysis of these products and provide some recommendations. But when we benchmark several software products, in addition to all the benefits commented, we obtain products that are continuously improved, recommendations for developers on the practices used and, from these practices, those that can be considered the best.

software evaluation in benchmarking Activities

To evaluate software is not a straightforward task; however, as this is an issue that has been thoroughly examined both in theory and practice, several authors have proposed different recommendations to consider (Basili et al., 1986; Perry et al., 2000; Freimut et al., 2002; Juristo & Moreno, 2001; Kitchenham et al., 2002).

These recommendations are also applicable to the software evaluations made during the benchmarking activities. However, when we have to define this kind of software evaluations, we must take into account some additional recommendations.

And the most important recommendation is that the evaluation of the benchmarking procedure must be an **improvement-oriented activity**. Its intended results will not only be used for comparing the different software products, but for learning how to improve such products. This requires that the evaluation yield not only some comparative results but also that it show the practices that produced these results.

Another recommendation is that benchmarking evaluations should be as **general** as possible since they will be performed by different groups of people in different locations and on different software.

Benchmarking is a process driven by a community; therefore, to gain credibility, effectiveness and impact, its evaluations should also be **community-driven**.

Additionally, benchmarking evaluations should be **reproducible** since they are intended to be used not only by the people participating in the benchmarking, but by the whole community. This requires that the evaluation be thoroughly detailed, providing public data and procedures.

To perform the evaluations consumes significant resources; these evaluations, on the other

hand, must be made by several groups of people. Therefore, evaluations should be as **economical** as possible, employing for this activity common evaluation frameworks and, when this is not possible, limiting the scope of the evaluation.

Furthermore, as benchmarking is a continuous process, benchmarking evaluations should have a **limited scope**, leaving other objectives for the next benchmarking iterations and incrementing progressively the complexity of the evaluations. We must add here that a broader evaluation scope does not entail better results but more resources.

As the next section shows, most of these recommendations should also be adopted in the benchmark suites. Therefore, it is advisable to **use benchmark suites** in the evaluations.

benchmark suites

A benchmark suite is a collection of benchmarks, being a benchmark *a test or set of tests used to compare the performance of alternative tools or techniques* (Sim et al., 2003).

A benchmark definition must include the following:

- The **context** of the benchmark, namely, which tools and which of their characteristics are measured with it (efficiency, interoperability, portability, usability, etc.).
- The **requirements** for running the benchmark, namely, the tools (hardware or software), data, or people needed.
- The **input variables** that affect the execution of the benchmark and the values that the variables will take.
- The **procedure** to execute the benchmark and obtain its results.
- The **evaluation criteria** followed to interpret these results.

A benchmark suite definition must include the definitions of all its benchmarks. Generally, all these benchmarks share some of their characteristics, such as context or requirements. These characteristics, therefore, must be specified at the benchmark suite level, and not individually for each benchmark.

Desirable Properties of a Benchmark Suite

The properties below, which are extracted from the works of different authors (Bull et al., 1999; Shirazi et al., 1999; Sim et al., 2003; Stefani et al., 2003), can help both to develop new benchmark suites and to assess the quality of the existing ones before being used.

Although a good benchmark suite should have most of these properties, each evaluation will require considering some of them previously. However, we must not forget that achieving these properties completely is not possible since the increment of some properties has a negative effect on the others.

- **Accessibility.** A benchmark suite must be accessible to anyone interested. This involves providing (a) the necessary software to execute the benchmark suite, (b) the software documentation, and (c) the software source code to increase transparency. Then the results obtained from executing the benchmark suite should be made public so that anybody can execute it and then compare his/her results with those available.
- **Affordability.** Using a benchmark suite normally entails a number of costs regarding human, software, and hardware resources. Thus, the costs of using a benchmark suite must be lower than those of defining, implementing, and carrying out any other experiments that fulfil the same goal. On the other hand, the resources consumed in the execution of a benchmark suite can be reduced by (a) automating the execution of the benchmark suite, (b) providing components for data collection and analysis, and (c) facilitating its use in different heterogeneous systems.

- **Simplicity.** The benchmark suite must be simple and interpretable and should be well documented; therefore, whoever wants to use it must be able to understand how it works and the results that it yields. If the benchmark suite is not transparent enough, its results will be questioned and may be interpreted incorrectly. To avoid this, the elements of the benchmark suite should have a common structure and use, and common inputs and outputs. Measurements, on the other hand, should have the same meanings across the benchmark suite.
- **Representativity.** The actions that perform the benchmarks composing the benchmark suite must be representative of the actions normally performed on the system.
- **Portability.** The benchmark suite should be executed on a wide range of environments and should be applicable to as many systems as possible. Besides, it should be specified at a high enough level of abstraction to ensure that it can be transferred to different tools and techniques and that is not biased against other technologies.
- **Scalability.** The benchmark suite should be parameterised to allow scaling the benchmarks with varying input rates. In addition, it should work with tools or techniques of different levels of maturity and should be applicable to research prototypes and commercial products.
- **Robustness.** The benchmark suite must allow for unpredictable environment behaviours and should not be sensitive to factors irrelevant to the study. When running the same benchmark suite on a given system and under the same conditions several times, the results obtained should not vary considerably.
- **Consensus.** The benchmark suite must be developed by experts capable of applying their knowledge of the domain and of identifying the key problems. Additionally, it should be assessed and agreed on by the whole community.

eVAlu At Ion And benchm Ark Ing w Ith In the sem Ant Ic web

This section provides an overview of the evaluation and benchmarking trends now occurring in the Semantic Web area; it also describes to what extent the evaluation and benchmarking activities performed on the Semantic Web technology can be partially or totally reused in different tools, and the facilities provided for doing so.

To this end, we have performed a survey of the main conferences on the Semantic Web field and of the workshops whose main topic is Semantic Web technology evaluation. We have examined the proceedings of five conferences: International Semantic Web Conference (ISWC), European Semantic Web Conference (ESWC), Asian Semantic Web Conference (ASWC), International Conference on Knowledge Capture (K-CAP), and International Conference on Knowledge Engineering and Knowledge Management (EKAW); and we have studied five workshops: Workshop on Evaluation of Ontology-based Tools (EON), Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS), Workshop on Practical and Scalable Semantic Systems (PSSS), International Workshop on Ontology Matching (OM), and Workshop on Integrating Ontologies (IntOnt). The survey includes all the papers accepted in these conferences and workshops from 2000 to 2006.

It is clear that the papers examined, which were presented in the conferences and workshops above commented, do not provide exhaustive information, but they can provide an overview of the current trends.

We consider that fulfilling the desirable properties of a benchmark suite and the recommendations for defining evaluations in benchmarking activities, both defined in the previous section, is

an indication of evaluation reusability. And thus, with these desirable properties and recommendations in mind we have produced a questionnaire that should be filled in for each of the selected papers. As an example, the questions asked for assessing the portability of an evaluation approach are the following:

- *In which type of tools can the evaluation be performed?*
- *Can the evaluation approach be transferred to other tools?*
- *On which of the operating systems/platforms can the evaluation be performed?*

Figure 1. Evaluation-related papers in conferences

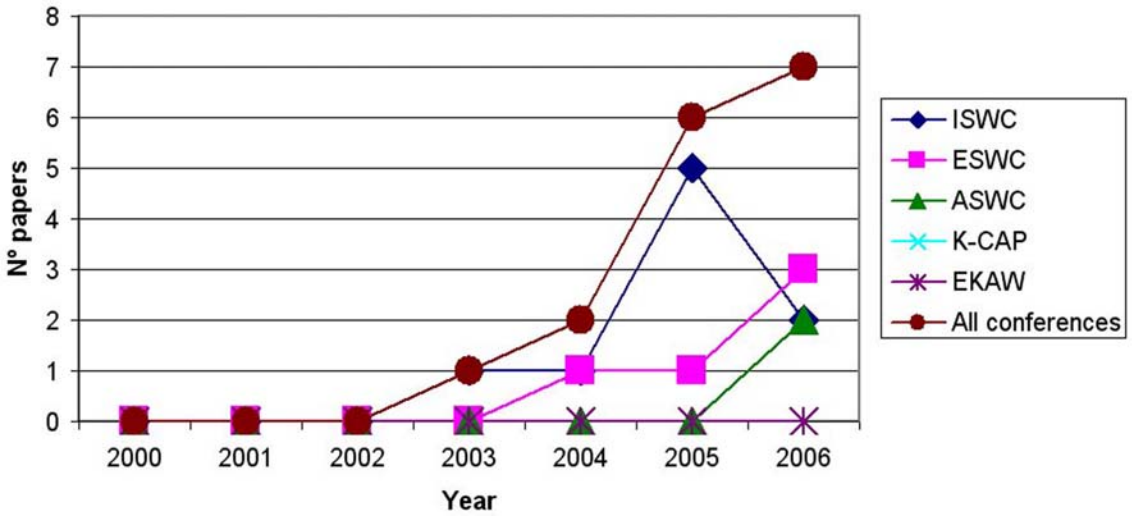
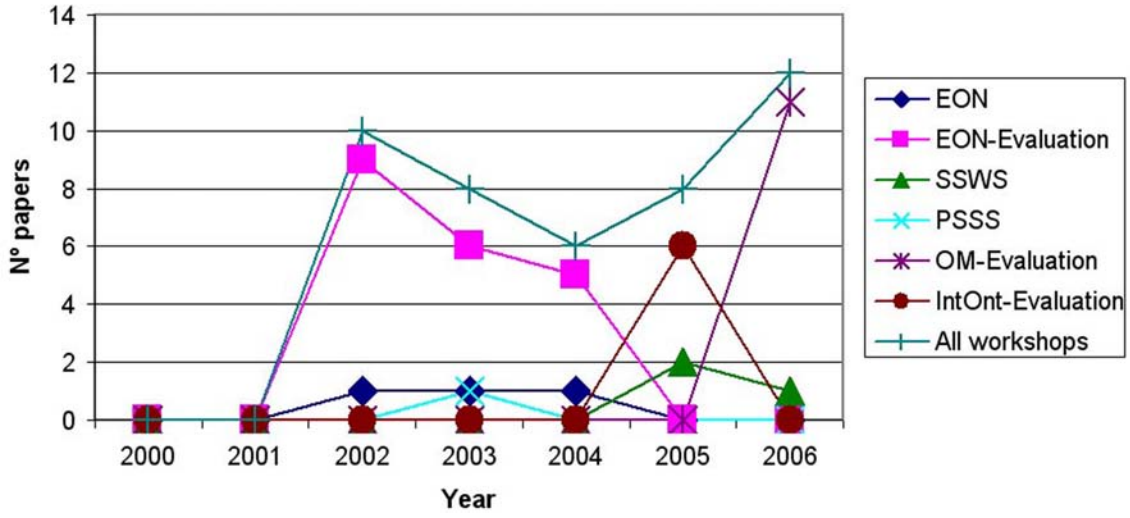


Figure 2. Evaluation-related papers in workshops



Although software evaluations are frequent in research papers, we focused on those papers where the evaluation approaches followed are reusable to a certain extent. However, we did not distinguish between evaluations that are performed within the benchmarking activities and evaluations that are not, nor did we distinguish between these benchmarking activities that use benchmark suites and those that do not. Thus, the criteria we followed to select the papers are:

- The paper describes how to evaluate several tools, or it shows the results of the evaluation over several tools, or both.
- The evaluation described in the paper involves more than one tool or is intended to be applied to more than one tool.
- The evaluation described in the paper is targeted to software tools or to methods implemented by some software.

With these criteria we selected 61 papers and filled in the questionnaire. Of these papers, 21 deal with the description and application of an evaluation, 7 simply describe an evaluation, and 33 show how an evaluation is made. Among the papers selected, we included those workshop papers that present proposals for performing a common evaluation and the papers written on this evaluation by the participants together with the results of their findings.

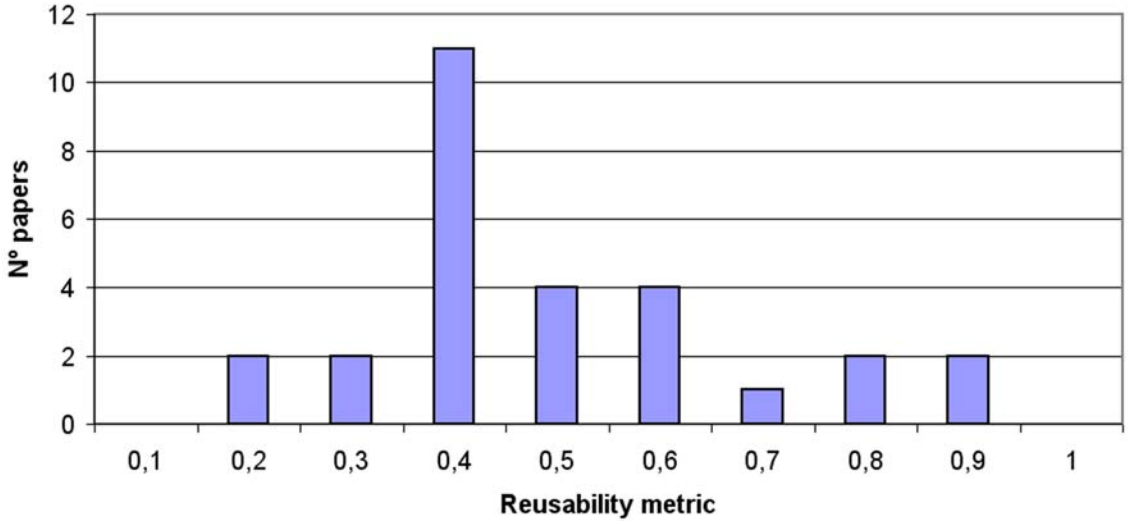
Figure 1 and Figure 2 show the trend concerning papers related to evaluation that were published in the last few years. As for the 17 papers presented in the conferences, they have led us to conclude, first, that the number of papers increases every year and, second, that no evaluation-related papers were submitted to the EKAW and K-CAP conferences; this may occur because these conferences are more oriented to knowledge management than to the Semantic Web. With regard to the 44 papers presented in workshops, we have noticed that only 7 of them are regular papers and that the other papers are either evaluation proposals or evaluation results. There is a call for participation in evaluations every year; we have observed that in these evaluations the number of evaluation contributions varies, whereas the number of regular papers is more or less constant.

Our survey shows the results of the reusability of the evaluation approaches in which it can be observed that some of them are positive and some, negative.

The positive results confirm that, in general, the evaluation approaches are easy to understand because they clearly establish both the input data, according to a common structure, and the evaluation criteria for analysing the results. In addition, in most cases, performing the evaluations is not expensive since the evaluation approaches provide common evaluation frameworks and, quite often, the whole evaluation or part of it can be automated. In other cases, however, some software supports the evaluation, being this software and its source code usually accessible. Scalability and robustness have also been taken into account throughout the evaluations.

The negative results show that most of the papers deal with a small group of Semantic Web tools (ontology alignment tools, ontology development tools and ontology repositories) and that, in general, evaluations are not applicable to other types of tools. Besides, the accessibility of these evaluations is not high, the procedure to perform the evaluation is not always clearly defined, and the input data is not always accessible. We can add that only in a few cases the papers provide a web page with information on the evaluation. The input data is not representative of the actions performed on the system and, occasionally, the evaluation has been developed and agreed on by a community. Furthermore, the existing evaluation approaches are not transferrable and the cost of performing the evaluation is never considered; most of the evaluations are defined as one-time activities. Only in a few cases improvement is a goal and, quite often, the practices that lead to the

Figure 3. Histogram of the reusability metric



results or improvement recommendations cannot be obtained directly from these results.

As we mentioned before, each of the questions of the questionnaire is related to one of the desirable properties of a benchmark suite. Therefore, we have described the reusability of the evaluation presented in each paper taking into account the number of questions answered positively for each of these properties. For each paper, we have calculated the percentage of positive answers for each dimension and the mean of all these percentages. The resulting value ranges from 0 (not reusable at all) to 1 (totally reusable). Even though this is not a robust metric, it gives us a hint of the reusability of the evaluations.

Figure 3 presents the histogram of the reusability metric used with the 28 papers that describe how to perform an evaluation over several tools. The horizontal axis represents the different ranges of the reusability metric, whereas the vertical axis represents the number of papers in each range. We can clearly observe that most of the papers have low values and are far from the ideal situation.

As a summary of this section we can conclude that although the number of evaluation and benchmarking activities is continuously increasing in the Semantic Web area, such number is still not good enough to ensure a high quality technology, and that the activities carried out involved just a few types of Semantic Web technologies. Consequently, the reusability of the evaluation approaches is not high enough, which is a hindrance for their use.

the benchm Ark Ing methodology For sem Ant Ic web technology

This section summarises the benchmarking methodology for Semantic Web technology developed by the authors in the Knowledge Web European Network of Excellence. A detailed description of this methodology can be found in (García-Castro et al., 2004).

The methodology has been inspired by works of quality improvement from different fields. The main inputs for this methodology were the benchmarking methodologies of the business management community and their notions of continuous improvement and best practices. We have also taken into account the evaluation and improvement

processes proposed in the Software Engineering area such as those cited in Section 2.1.

benchmarking Actors

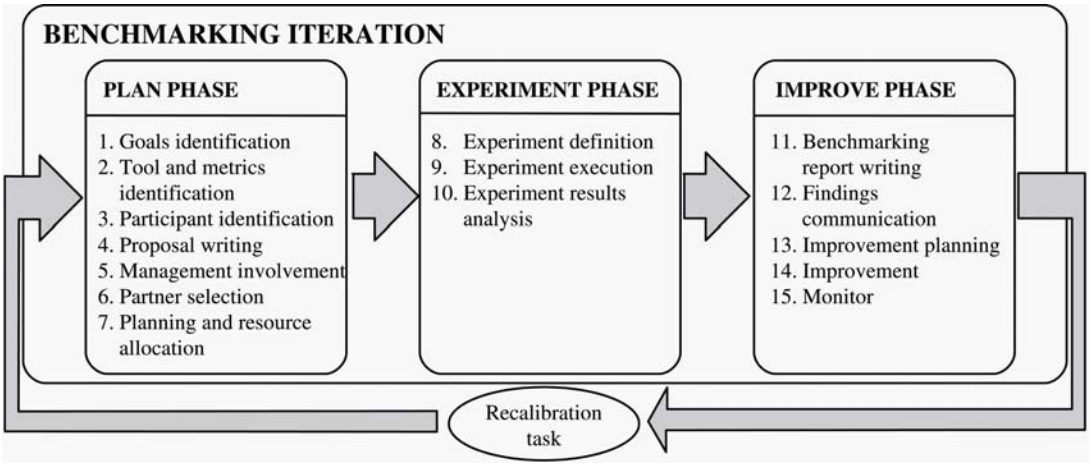
The tasks of the benchmarking process are carried out by different actors according to the kind of roles to be performed in each task. The different types of actors involved are the following:

- **Benchmarking initiator.** The benchmarking initiator is the member (or members) of an organisation who makes the first tasks of the benchmarking process. His/her work consists in preparing a proposal for carrying out the benchmarking activity in the organisation and in obtaining the management approval to perform it.
- **Organisation management.** The organisation management plays a key role in the benchmarking process: this actor must approve the benchmarking activity and the changes that result from it. The organization management must also assign resources to the benchmarking and integrate the benchmarking planning into the organisation planning.
- **Benchmarking team.** Once the organisation management approves the benchmarking proposal, the benchmarking team is composed. Their members have to belong to the organisation and are responsible for performing most of the remaining benchmarking processes.
- **Benchmarking partners.** The benchmarking partners are the organisations participating in the benchmarking activity. All the partners must agree on the steps to follow during the benchmarking, and their needs must be considered.
- **Tool developers.** The developers of the tool must implement the necessary changes in the tool to improve it. Some of the developers may also be part of the benchmarking team and, if so, care must be taken to minimise bias.

benchmarking process

The benchmarking methodology for Semantic Web technology describes a benchmarking process together with the main phases to follow when benchmarking this technology and it

Figure 4. The software benchmarking methodology



also provides a set of guidelines. Therefore, this methodology has a twofold use: to help carry out software benchmarking activities, and to know, at a specific time, which is the actual progress of a benchmarking activity.

The benchmarking process defined in this methodology is composed of a benchmarking iteration that is repeated forever. Each iteration, as shown in Figure 4, is composed of three phases (*Plan*, *Experiment* and *Improve*) and ends with a *Recalibration* task.

Plan Phase

The *Plan* phase is composed of the different tasks that must be performed (1) to prepare the benchmarking proposal, (2) to obtain support from the organisation management, (3) to find other organisations willing to participate in benchmarking, and 4) to plan benchmarking. These tasks are the following:

P1. Goals identification. In this task, the *benchmarking initiator* (the member or members of the organisation who are aware of the need for benchmarking) must identify the benchmarking goals according to the goals and strategies of the organization as well as the benefits and costs involved in carrying out benchmarking. However, every organization may have its own goals and these can be quite different. For example, some may be interested in assessing the performance and improving the quality of the software over its lifetime, others, in comparing their software with the software that is considered the best, whereas some others are interested in establishing or creating standards by analysing the different existing software.

P2. Software and metrics identification. In this task, the *benchmarking initiator* should make an analysis of the software products developed in the organisation in order to understand and document them, identifying the weaknesses and functionalities that require improvement. Then, he/she must select the products to be benchmarked, the functionalities relevant to the study and the evaluation criteria to follow to assess these functionalities; these criteria must take into account the organisation's software, the benchmarking goals, the benefits and costs identified in the previous task as well as other factors considered critical by the organisation, such as quality requirements, end-user needs, etc.

P3. Participant identification. In this task, the *benchmarking initiator* must identify and contact the members concerned with the software and the functionalities selected (managers, developers, end users, etc.) and other relevant participants that do not belong to the organisation (customers or consultants). The benchmarking initiator is responsible for organizing the benchmarking team and, quite often, he is a member of the team. The team should be small and include those organisation members whose work and interest are related to the software, who have a thorough understanding of the software and have gained valuable experience with it. They must be aware of the time they will spend in the benchmarking activity and of their responsibilities, and they should be trained in the tasks they will have to perform.

P4. Proposal writing. In this task, the *benchmarking team* (and the *benchmarking initiator*, if he does not belong to the team) must write a document with the benchmarking proposal. The proposal will be used as a reference along the benchmarking process and must include all the relevant information on the process: the information identified in the previous benchmarking tasks (goals, benefits, costs, software, metrics, members involved, and benchmarking team), a description of the benchmarking process, and a full detailed description of the benchmarking costs along with the resources needed. To do this, the benchmarking team should take into consideration the different intended readers of the benchmarking proposal, namely, the organisation managers, the

organisation developers, the members of partner organisations, and the members of the benchmarking team.

P5. Management involvement. In this task, the *benchmarking initiator* must bring the benchmarking proposal to the *organisation management*. This task is of great significance because the management approval is required if we want to continue with the benchmarking process. Management support will also be requested in the future, when the changes required for benchmarking will have to be implemented, either in the software or in the organisation processes that affect the software.

P6. Partner selection. In this task, the *benchmarking team* must collect and analyse information on the software products that are to be compared with the software selected, and on the organisations that develop the products. The benchmarking team must also select the software employed in the benchmarking study taking into account its relevance and use in the community or in the industry, its public availability, how the software has adopted the latest technological tendencies, etc. In order to obtain better results with benchmarking, the software selected should be the software considered the best. Then, the benchmarking team must contact the people from the organisations that develop these software products to find out whether they are interested in becoming *benchmarking partners*. These partners will also have to establish a team and to take the proposal to their own organisation management for approval. During this task, the benchmarking proposal will be modified by incorporating the partners' opinions and requirements. This will result in an updated proposal that, depending on the magnitude of the modifications, should be presented again to each organisation management for approval.

P7. Planning and resource allocation. In this task, the *organisation managements* and the *benchmarking teams* must specify the planning of the remainder of the process, considering the different resources that will be devoted to it, and finally they must reach a consensus. This planning must be given careful consideration and should be integrated into each organisation planning.

Experiment Phase

The *Experiment* phase is composed of the tasks in which the experimentation on the software products is performed. These tasks are the following:

E1. Experiment definition. In this task, the *benchmarking teams* of each organization must establish the experiment that will be performed on each of the software products, and then the members must agree on it. The experiment must be defined according to the benchmarking goals, the software functionalities selected, and their corresponding criteria, as stated in the benchmarking proposal. The experiment must also provide objective and reliable software data not just of its performance, but also of the reasons of its performance; in addition, its future reusability must be also be considered. The benchmarking teams must determine and agree on the planning to follow during the experimentation; this new planning must be decided according to the benchmarking planning established previously.

E2. Experiment execution. As indicated in the experimentation planning, explained in the previous task, the *benchmarking teams* must perform the established experiments on their software products. Then the data obtained from all the experiments must be compiled, documented, and written in a common format to facilitate its future analysis.

E3. Experiment results analysis. In this task, the *benchmarking teams* must analyse the results, detect and document any significant differences found in them, and determine the practices leading to these different results in order to identify whether, among the practices found, some of them can be considered the best practices. Then, the benchmarking teams should write a report with

all the findings of the experimentation, that is, the experimentation results, the differences in the results, the practices and the best practices found, etc.

Improve Phase

The *Improve* phase comprises the tasks where the results of the benchmarking process are produced and then communicated to the benchmarking partners; it also comprises the tasks where, in several cycles, the improvement of the different software products takes place. The tasks are the following:

I1. Benchmarking report writing. In this task, the *benchmarking teams* must write the benchmarking report. This report is intended to provide an understandable summary of the benchmarking carried out, and it should be written bearing in mind its different audiences: managers, benchmarking teams, developers, etc. The benchmarking report must include a) an explanation of the process followed, together with all the relevant information of the updated version of the benchmarking proposal; and b) the results and conclusions of the experiments presented in the experiment report, including the practices found and highlighting the best practices. The report should also contain the recommendations provided by the benchmarking teams for improving the software products according to the experiment results, the practices found, and the best practices implemented by the community.

I2. Findings communication. Here, the *benchmarking teams* must communicate, in successive meetings, the results of the benchmarking to their organisations and, particularly, to all the members concerned and identified when planning benchmarking. The goals of these meetings are:

- To obtain feedback from the members concerned on the benchmarking process, the results, and the improvement recommendations.
- To obtain support and commitment from the organisation members for implementing the improvement recommendations on the software.

Any feedback received during these communications must be collected, documented and analysed. This analysis may finally involve having to review the work done and to update the benchmarking report.

I3. Improvement planning. The last three tasks of the *Improve* phase (*Improvement planning*, *Improvement* and *Monitor*) form a cycle that must be carried out separately in each organisation. The *benchmarking teams* and the *organisation managements* must identify, from the benchmarking report and the monitoring reports, which are the changes needed to improve their software products. Besides, they must forecast the improvements to be achieved after performing these changes. Both the organisation management and the benchmarking team must provide mechanisms for ensuring improvements in their organisation and for measuring the software functionalities. These last mechanisms can be obtained from the *Experiment* phase. Then, the organisation management and the benchmarking team must establish a planning for improving the software benchmarked, taking into account the different resources devoted to the improvement. This planning must be then integrated into the organisation planning.

I4. Improvement. It is in this task where the *developers* of each of the software product must implement the necessary changes to achieve the desired results. For this, they must measure the state of the software before and after implementing any changes, using for that purpose the measurement mechanisms provided by the benchmarking team in the previous task. Then, the developers must compare the resulting measurements with those that were obtained before implementing the changes and with the improvement forecasted.

I5. Monitor. In each organisation, the *benchmarking team* must provide *software developers* with means for monitoring the organisation's software. Software developers must periodically monitor the software and write a report with the results of this process. These results may show the need for new improvements in software and may also mean the beginning of a new improvement cycle which involves having to perform again the two tasks previously mentioned: *Improvement Planning* and *Improvement*.

Recalibration Task

The recalibration task is performed at the end of each iteration. Here the *benchmarking teams* must recalibrate the process by applying the lessons learnt while performing it. Thus, the organisation (and the whole community) achieves improvement not just in the software, but also in the benchmarking process. This recalibration is needed because both the software and the organisations evolve and innovate over time.

benchmarking the Interoperability of Ontology development tools using rdf(s) as the Interchange Language

This section presents how we have applied the software benchmarking methodology, presented in the previous section, to one important problem of the Semantic Web: technology interoperability.

Ontologies permit interoperability among heterogeneous Semantic Web technologies, and ideally, one could use all the existing technologies seamlessly. The technologies appear in different forms (ontology development tools, ontology repositories, ontology alignment tools, reasoners, etc.), but although all these tools use different kinds of ontologies, not all of them share a common knowledge representation model.

This diversity in the representation formalisms of the tools causes problems when the tools try to interoperate. This is so because the tools require translating their ontologies from their own knowledge models to a common model and vice versa, and these problems occur even when we employ standard APIs for managing ontologies in the common knowledge model.

As we have observed in previous workshops on Evaluation of Ontology-based Tools (EON) (Sure & Corcho, 2003), interoperability among different ontology tools is not straightforward. Finding out why interoperability fails is cumbersome and not at all trivial because any assumption made for translating ontologies within one tool may easily prevent the successful interoperability with other tools.

To solve this problem, the Knowledge Web European Network of Excellence organized a benchmarking of interoperability of ontology development tools using RDF(S) as the interchange language. Its goal was to assess and improve the interoperability of the tools.

The section that follows describes such benchmarking activity. The methodology presented in the previous section provides the general guidelines that can be adapted to this case. So we present here how this new benchmarking was organized, the experiments conducted on the participating tools, and its results.

organising the benchmarking

The goals of benchmarking the interoperability of ontology development tools are related to the benefits pursued through it, and these are:

- To *evaluate and improve* their interoperability.
- To acquire a *deep understanding* of the practices used to develop the importers and exporters of these tools, and to extract from these practices those that can be considered the *best practices*.

Table 1. Ontology tools participating in the benchmarking

Tool	Knowledge model	Version	Developer	Experimenter
Corese	RDF(S)	2.1.2	INRIA	INRIA
Jena	RDF(S)	2.3	HP	U. P. Madrid
KAON	RDF(S) extension	1.2.9	U. Karlsruhe	U. Karlsruhe
Sesame	RDF(S)	2.0 alpha 3	Aduna	U. P. Madrid
Protégé	Frames	3.2 beta build 230	Stanford U.	U. P. Madrid
WebODE	Frames	2.0 build 109	U. P. Madrid	U. P. Madrid

- To produce *recommendations* on their interoperability for users.
- To create *consensual processes* for evaluating their interoperability.

These goals concern different communities that somehow are related to the ontology development tools, namely, the research community, the industrial community, and the tool developers.

Participation in the benchmarking was open to any organisation irrespective of being a Knowledge Web partner or not. To involve other organisations in the process with the aim of having the best-in-class tools participating, several actions were taken:

- The benchmarking proposal, the document being used as a reference along the benchmarking, was published as a public web page[b] and included all the relevant information about the benchmarking: motivation, goals, benefits and costs, tools and people involved, planning, related events, and a complete description of the experimentation and the benchmark suites.
- Research was carried out on the existing ontology development tools, both the freely available and the commercial versions, which could export and import to and from RDF(S); In addition, their developers were contacted. Any tool capable of importing and exporting RDF could participate in the benchmarking or will benefit from the created benchmarks in a near future.
- The interoperability benchmarking was announced with a call for participation through the main mailing lists of the Semantic Web area and through lists specific to ontology development tools.

Six tools took part in the benchmarking, three of which are ontology development tools: KAON[c], Protégé[d] (using its RDF backend), and WebODE[e]; the other three are RDF repositories: Corese[f], Jena[g] and Sesame[h]. As Table 1 shows, the tools do not share a common knowledge model and benchmarking was not always performed by the tool developers.

The experimentation conducted on the tools aimed to obtain results for interoperability improvement. Therefore, other quality attributes such as performance, scalability, interoperability, robustness, etc. were not considered. However, an approach for benchmarking the performance and scalability of ontology development tools can be found in (García-Castro & Gómez-Pérez, 2005).

The experimentation was carried out taking into account the most common ways of interchanging ontologies that ontology tools provide, such as the following:

- Interchanging ontologies by exporting them from a tool into an interchange language and then importing them into the other tool.

- Using RDF(S) as the interchange language, and serializing the ontologies into the RDF/XML syntax. A future benchmarking activity inside Knowledge Web will cover the case of using OWL as the interchange language.

The interoperability of ontology tools using an interchange language depends on the capabilities of the tools to import and export ontologies from/to this language. Therefore, the experimentation included not only the evaluation of the interoperability but also of the RDF(S) import and export functionalities.

The evaluation criteria must describe in depth the import, export and interoperability capabilities of the tools, whereas the experiments to be performed in the benchmarking must provide data explaining how the tools comply with these criteria. Therefore, to obtain detailed information about these capabilities, we need to know:

- The elements of the *internal knowledge model* of an ontology development tool that can be imported from RDF(S), exported to RDF(S) and interchanged with other tool, using RDF(S) as the interchange language.
- The *secondary effects* of importing, exporting, and interchanging these components, such as insertion or loss of information.
- The *subset of elements* of the internal knowledge models that these tools may use to interoperate correctly.

To obtain these experimentation data, we defined three benchmark suites that evaluate the capabilities of the tools (García-Castro et al., 2006), which were common for all the tools. Since the quality of the benchmark suites to be used is essential for the results, the first step was to agree on the definition of the suites. Then, we decided to make the import and export experiments before the interoperability one because the results of the first experiments affected those of the second.

A benchmark execution comprises (a) the definition of the expected ontology that results from importing, exporting or interchanging the ontology described in the benchmark, (b) the import, export, or interchange of the ontology defined in the benchmark, and (c) the comparison of the expected ontology with the imported, exported or interchanged ontology, checking whether there is some addition or loss of information. The steps to follow to execute the three benchmark suites are similar.

The benchmark suites were intended to be executed manually but, as they contained many benchmarks, it was highly recommended to execute them (or part of them) automatically. In the cases of Corese, Jena, Sesame, and WebODE, most of the experiment was automated. In the other cases, it was performed manually.

The benchmarking web page[i] contains the results of the experiments and a complete and detailed description of (a) the benchmark suites, (b) all the files to be used in the experiments, and (c) the templates for collecting the results.

benchmark suites

The benchmark suites check the correct import, export and interchange of ontologies that model a simple combination of ontology components (classes, properties, instances, etc.). Because one of the goals of benchmarking is to improve the tools, the benchmark ontologies are kept simple on purpose in order to isolate the causes of the problems and to identify possible problems.

As the ontology tools that participated in benchmarking had different internal knowledge models, both the experimentation and the analysis of the results were based on a common group of ontology modelling primitives, available both in RDF(S) and in these tools. On the other hand, covering this common group exhaustively would

yield a huge number of benchmarks; so we only considered the components most widely used for modelling ontologies in ontology development tools: classes, instances, properties with domain and range, literals, and class and property hierarchies. The remainder of the components has not been dealt with so far.

The **RDF(S) Import Benchmark Suite** contains 82 benchmarks[j], which define a simple RDF(S) ontology serialized in a RDF/XML file, which must be loaded into the ontology development tool.

In order to isolate the factors that affect the correct import of an ontology, we defined two types of import benchmarks: one that evaluates the import of the different combinations of components of the RDF(S) knowledge model, and the other type that evaluates the import of the different variants of the RDF/XML syntax, as stated in the RDF/XML specification.

Table 2 shows the categories of the RDF(S) Import Benchmark Suite, the number of benchmarks, and the components used. All the RDF(S) files to be imported can be downloaded from a single file; besides, templates are provided for collecting the execution results.

The **RDF(S) Export Benchmark Suite** comprises 66 benchmarks[k], which describe an ontology that must be modelled in the tool and saved to a RDF(S) file.

We have defined two types of benchmarks for isolating the two factors that affect the correct export of an ontology: one type evaluates the correct export of the combinations of components of the ontology development tool knowledge model, and the other evaluates the export of ontologies using concepts and properties whose names have characters restricted by RDF(S), such as those characters that are forbidden when representing RDF(S) or XML URIs.

Table 3 shows the categories of the benchmark suite. The table contains the number of benchmarks and the components used in each category. Templates are also provided for collecting the execution results.

Since the factors that affect both the correct interchange of an ontology (besides the correct functioning of the importers and exporters) and the knowledge model (used for defining the ontologies) are the same as those that affect the RDF(S) Export Benchmark Suite, the ontologies described in the RDF(S) Interoperability Bench-

Table 2. Categories of the import benchmarks

Category	No.	Components used
Class	2	*rdfs:Class*
Metaclass	5	*rdfs:Class*, *rdf:type*
Subclass	5	*rdfs:Class*, *rdfs:subClassOf*
Class and property	6	*rdfs:Class*, *rdf:Property*, *rdfs:Literal*
Property	2	*rdf:Property*
Subproperty	5	*rdf:Property*, *rdfs:subPropertyOf*
Property with domain and range	24	*rdfs:Class*, *rdf:Property*, *rdfs:Literal*, *rdfs:domain*, *rdfs:range*
Instance	4	*rdfs:Class*, *rdf:type*
Instance and property	14	*rdfs:Class*, *rdf:type*, *rdf:Property*, *rdfs:Literal*
Syntax and abbreviation	15	*rdfs:Class*, *rdf:type*, *rdf:Property*, *rdfs:Literal*

Table 3. Categories of the export benchmarks

Category	No.	Components used
Class	2	*class*
Metaclass	5	*class*, *instanceOf*
Subclass	5	*class*, *subClassOf*
Class and object property	4	*class*, *object property*
Class and datatype property	2	*class*, *datatype property*, *literal*
Object property	14	*object property*
Datatype property	12	*datatype property*
Instance	4	*class*, *instanceOf*
Instance and object property	9	*class*, *instanceOf*, *object property*
Instance and datatype property	5	*class*, *instanceOf*, *datatype property*, *literal*
URI character restrictions	4	*class*, *instanceOf*, *object property*, *datatype property*, *literal*

mark Suite are identical to those of the RDF(S) Export Benchmark Suite.

The evaluation criteria are common for the three benchmark suites, and are defined as follows:

- **Modelling** (*YES*/*NO*). The ontology tool can model the ontology components described in the benchmark.
- **Execution** (*OK*/*FAIL*). The execution of the benchmark is normally carried out seamlessly, and the tool always produces the expected result. However, when an execution fails, the following information is required:
  - The causes of the failure.
  - The changes performed if the tool had been previously corrected to pass a benchmark correctly.
- **Information added or lost.** The information added to or lost in the ontology interchange when executing the benchmark.

In the export and interoperability benchmark suites, if a benchmark describes an ontology that cannot be modelled in a certain tool, such benchmark cannot be executed in the tool, being the *Execution* result *N.E.* (Non Executed). However, in the import benchmark suites, even if a tool cannot model some components of the ontology, it should be able to import the rest of the components correctly.

Import and export results

The results obtained when importing from and exporting to RDF(S) depend mainly on the knowledge model of the tool that executes the benchmark suite. The tools that natively support the RDF(S) knowledge model (Corese, Jena and Sesame, essentially the RDF repositories) do not need to perform any translation in the ontologies when importing/exporting them from/to RDF(S). The RDF repositories import and export correctly all the combinations of components from/to RDF(S) because these operations do not require any translation.

In the case of tools with non-RDF knowledge models (KAON, Protégé and WebODE, the ontology development tools), some of their knowledge model components can also be represented in RDF(S), but some others cannot, and these tools do need to translate ontologies between their

Table 4. Combinations of components modelled by the tools

Combination of components	RDF repos.	KAON	Protégé	WebODE
Classes	Y	Y	Y	Y
...instance of metaclasses	Y	Y	Y	N
Class hierarchies without cycles	Y	Y	Y	Y
...with cycles	Y	N	N	N
Datatype properties without domain or range	Y	Y	Y	N
...with multiple domains	Y	Y	N	N
...whose range is String	Y	Y	Y	Y
...whose range is a XML Schema datatype	Y	Y	N	Y
Object properties without domain or range	Y	Y	Y	N
...with multiple domains or ranges	Y	Y	N	N
...with a domain and range	Y	Y	Y	Y
Instances of a single class	Y	Y	Y	Y
...of multiple classes	Y	Y	Y	N
...related via object or datatype properties	Y	Y	Y	Y
...related via datatype properties whose range is a XML Schema datatype	Y	N	N	Y

knowledge models and RDF(S). Finally, we must add that not all the combinations of components of the RDF(S) knowledge model that have been considered can be modelled into all the tools, as Table 4 shows.

Next, we present an analysis of the results of importing and exporting in the ontology development tools that participated in the benchmarking.

Import Results

In general, the ontology development tools import correctly from RDF(S) all or most of the combinations of components that they model; they seldom add or lose information. The only exceptions are:

- Protégé, which presents problems, but only when it imports classes or instances that are instances of multiple classes.
- WebODE, which presents problems, but only when it imports properties with a XML Schema datatype as range.

When the tools import ontologies with combinations of components that they cannot model, they lose the information about these components. Nevertheless, these tools usually try to represent such components partially using for this other components from their knowledge models. In most cases, the importing is performed correctly. The only exceptions are:

- KAON, which causes problems when it imports class hierarchies with cycles.
- Protégé, which causes problems when it imports class and property hierarchies with cycles and properties with multiple domains.
- WebODE, which causes problems when it imports properties with multiple domains or ranges.

When dealing with the different variants of the RDF/XML syntax, we can observe that the ontology development tools

- Import correctly resources with the different URI reference syntaxes.
- Import correctly resources with different syntaxes (shortened and unshortened) of empty nodes, of multiple properties, of typed nodes, of string literals, and of blank nodes. The only exceptions are: KAON when it imports resources with multiple properties in the unshortened syntax; and Protégé when it imports resources with empty and blank nodes in the unshortened syntax. Do not import language identification attributes (*xml:lang*) in tags.

Export Results

In general, the ontology development tools export correctly to RDF(S) all or most of the combinations of components that they model with no loss of information. In particular:

- KAON causes problems only when it exports to RDF(S) datatype properties without range and datatype properties with multiple domains plus a XML Schema datatype as range.
- Protégé causes problems only when it exports to RDF(S) classes or instances that are instances of multiple classes and template slots with multiple domains.

When ontology development tools export components present in their knowledge model that cannot be represented in RDF(S), such as their own datatypes, they usually insert new information in the ontology, but they also lose some.

When dealing with concepts and properties whose names do not fulfil URI character restrictions, each ontology development tool behaves differently:

- When names do not start with a letter or "_", some tools leave the name unchanged, whereas others replace the first character with "_".
- Spaces in names are replaced by "-" or "_", depending on the tool.
- URI reserved characters and XML delimiter characters are left unchanged, replaced by "_", or encoded, depending on the tool.

Interoperability r esults

The RDF repositories (Corese, Jena and Sesame) interoperate correctly between themselves, because they always import and export from/to RDF(S) correctly. This produces that the interoperability between the ontology development tools and the RDF repositories depends only on the capabilities of the former to import and export from/to RDF(S); therefore, the results of this interoperability are identical to those presented in the previous section.

The import and export results presented in previous sections indicate that some problems arise in the process of importing and exporting ontologies, whereas the interoperability results, on the other hand, show more problems.

As a general comment we can say that interoperability between the tools depends on

a. the correct functioning of their RDF(S) importers and exporters and
b. the way chosen for serializing the exported ontologies in the RDF/XML syntax.

Furthermore, we have observed that the problems affecting any of these factors also affect the results of not just one but several benchmarks. This means that, in some cases, to correct a single import or export problem, or to change the way of serializing ontologies can produce significant interoperability improvements.

Below we list the components that can be interchanged between the tools. These components are

Table 5. Components interchanged between the tools

Combination of components	K-K	P-P	W-W	K-P	K-W	P-W	K-P-W
Classes	Y	Y	Y	Y	Y	Y	Y
...instance of a single metaclass	Y	Y	-	N	-	-	-
...instance of a multiple metaclasses	Y	N	-	N	-	-	-
Class hierarchies without cycles	Y	Y	Y	Y	Y	Y	Y
Datatype properties without domain or range	Y	Y	-	N	-	-	-
...with multiple domains	Y	-	-	-	-	-	-
...whose range is String	Y	Y	Y	N	N	Y	N
...whose range is a XML Schema datatype	Y	-	Y	-	Y	-	-
Object properties without domain or range	Y	Y	-	Y	-	-	-
...with multiple domains or ranges	Y	-	-	-	-	-	-
...with a domain and range	Y	Y	Y	Y	Y	Y	Y
Instances of a single class	Y	Y	Y	Y	Y	Y	Y
...of multiple classes	Y	N	-	N	-	-	-
...related via object properties	Y	Y	Y	Y	Y	Y	Y
...related via datatype properties	Y	Y	Y	N	Y	Y	N
...related via datatype properties whose range is a XML Schema datatype	-	-	Y	-	-	-	-

summarized in Table 5; each column of the table shows whether the combination of components can be interchanged between a group of tools[l]. The "-" character means that the component cannot be modelled in some of the tools and, therefore, cannot be interchanged between them.

Interoperability Using the Same Tool

Ontology development tools seem to pose no problems when the source and the destination of an ontology interchange are the same tool. The only exception is Protégé when it interchanges classes that are instances of multiple metaclasses and instances of multiple classes; this is so because Protégé does not import resources that are instances of multiple metaclasses.

Interoperability between Each Pair of Tools

The interoperability between different tools varies depending on the tools. As the detailed interoperability results show, in some cases, the tools are able to interchange certain components from one tool to another, but not the other way round.

When **KAON** interoperates with **Protégé**, they can interchange correctly some of the common components that these tools are able to model. However, with components such as classes that are instances of a single metaclass or of multiple metaclasses, datatype properties without domain or range, datatype properties whose range is *String*, instances of multiple classes, and instances related through datatype properties, we have encountered some problems.

When **KAON** interoperates with **WebODE**, they can interchange correctly most of the com-

mon components that these tools can model, but when they interchange datatype properties with domain and whose range is *String,* the results are not the same.

When **Protégé** interoperates with **WebODE**, they can interchange correctly all the common components that these tools can model.

Interoperability between All the Tools

Interoperability between **KAON**, **Protégé** and **WebODE** can be achieved by most of the common components that these tools can model. The only components that these tools cannot use are datatype properties with domain and whose range is *String,* and instances related through datatype properties.

Therefore, interoperability was achieved among the tools that participated in the benchmarking by using classes, class hierarchies without cycles, object properties with domain and with range, instances of a single class, and instances related through object properties.

Interoperability Regarding URI Character Restrictions

Interoperability is low when tools interchange ontologies containing URI character restrictions in class and property names. This is so because tools usually encode some or all the characters that do not comply with these restrictions, which provokes changes in class and property names.

r ecommendations

Recommendations for Ontology Engineers

This section offers recommendations for ontology engineers which use more than one ontology tool to build ontologies. Depending on the tools used, the level of interoperability may be higher or lower, as can be seen in Section 5.4.

If the ontology is being developed bearing in mind interoperability, ontology engineers should be aware of the components that can be represented in the ontology development tools and in RDF(S). And they should try to use the common knowledge components that these tools have so as to avoid the knowledge losses commented above.

Ontology engineers should also be aware of the semantic equivalences and differences between the knowledge models of the tools and the interchange language. For example, in Protégé, multiple domains in template slots are considered the union of all the domains, whereas in RDF(S) multiple domains in properties are considered the intersection of all the domains; in WebODE, on the other hand, instance attributes are local to a single concept, whereas in RDF(S) properties are global and can be used in any class.

It is not recommended to name resources using spaces or any character that is restricted in the RDF(S), URI, or XML specifications.

When the RDF repositories interoperate, even though these repositories export and import correctly to RDF(S), ontology engineers should consider the limitations that other tools have when they export their ontologies to RDF(S).

Recommendations for Tool Developers

This section includes general recommendations for improving the interoperability of the tools while developing them. In (García-Castro et al., 2006), we offer full detailed recommendations regarding the results and practices gathered to improve each of the participant tools. Although it is not compulsory to follow these recommendations, they help correct interoperability problems as we could observe when we analysed the results.

The interoperability between ontology tools (using RDF(S) as the interchange language) depends on how the importers and exporters of these tools work; on the other hand, how these importers

and exporters work depends on the development decisions made by the tool developers, and these are different people with different needs. Therefore, to provide general recommendations for developers is not straightforward, though some comments can be extracted from the analysis of the benchmarking results.

In some occasions, a development decision will produce interoperability improvement with some tools and interoperability loss with others. For example, when exporting classes that are instances of a metaclass, some tools require that the class be defined as instance of *rdfs:Class*, whereas other tools require the opposite.

Tool developers, therefore, should analyze the collateral consequences of the development decisions. Thus, if a datatype is imported as a class in the ontology, then the literal values of this datatype should be imported as instances in the ontology, which would complicate the management of these values.

They also should be aware of the semantic equivalences and differences between the knowledge models of their tool and the interchange language; on the other hand, the tools should notify the user when the semantics is changed.

The first requirement for achieving interoperability is that the importers and exporters of the tools be robust and work correctly when dealing with unexpected inputs. Although this is an obvious comment, the results show that this requirement is not achieved by the tools and that some tools even crash when importing some combinations of components.

Above all, tools should deal correctly with the combinations of components that are present in the interchange language but cannot be modelled in them. For example, although cycles in class and property hierarchies cannot be modelled in some ontology development tools, these tools should be able to import these hierarchies by eliminating the cycles.

If developers want to export components that are commonly used by ontology development tools, the components should be completely defined in the file. This means that metaclasses and classes in class hierarchies should be defined as instances of *rdfs:Class*, properties should be defined as instances of *rdf:Property*, etc.

Exporting complete definitions of other components can cause problems if these are imported by other tools. And not every tool deals with datatypes defined as instances of *rdfs:Datatype* in the file, or with *rdf:datatype* attributes in properties.

If the document does not define a default namespace, every exported resource should have a namespace.

conclus Ion

This chapter states the need to evaluate and benchmark the Semantic Web technology and provides some references that can be helpful in these activities. It also presents the authors' approach to software benchmarking and compares it with other existing evaluation and benchmarking approaches.

We have tried to explain how the benchmarking methodology can help assess and improve software, whereas the use of benchmark suites is advisable when performing evaluations in benchmarking.

One of the strong points we make on benchmarking is its community-driven approach. Benchmarking should be performed by the experts of the community since the benefits obtained after performing it affect the whole community.

Benchmarking does not imply comparing the results of the tools but comparing the practices that lead to these results. Therefore, experimentation should be designed to obtain these practices as well as the results.

However, as we have seen, benchmarking is not the solution to every case. In a preliminary step, developers would have to asses whether benchmarking is the correct approach; as bench-

marking is useful when the goals are to improve the software and to extract the practices performed by others.

Benchmarking is an activity that takes long time to perform because it requires tasks that are not immediate: announcements, agreements, etc. Therefore, benchmarking activities should start early in time, and the benchmarking planning should consider a realistic duration of the benchmarking and the resources needed for carrying them out.

We have also shown how we have applied the benchmarking methodology to a concrete case in the Semantic Web area: interoperability of ontology development tools using RDF(S) as interchange language.

Providing benchmark suites in the benchmarking allows evaluating other tools with RDF(S) import and export capabilities without their having to participate in the benchmarking; this can be useful both while the tools are being developed and afterwards, when their development has finished. In addition, the benchmarking results can be used by ontology development tool users that may find problems when interchanging ontologies or may want to foresee the results of a future interchange.

Although it is not required that the tool developers participate in the benchmarking and perform the experiments over their tool, their involvement facilitates the execution and analysis of the experimentation results to a large extent. In all the cases where tool developers carried out the experimentation over their own tools, a great improvement occurred before the *Improve* phase of the methodology because developers were able to detect problems and correct their tools while executing the benchmark suites.

We have observed that the manual execution of the experiments and the analysis of the results cause the benchmark suite to be costly. Consequently, tool developers often automate the execution of the benchmark suites, but not always. Another drawback of the manual execution of experiments is that the results obtained depend on the people performing these experiments, on their expertise with the tools, and on their ability to extract the practices performed.

Future rese Arch dIrect Ions

As shown in Section 3, current evaluation and benchmarking activities over the Semantic Web technology are scarce and a hindrance to the full development and maturity of this technology. The Semantic Web needs to produce methods and tools for evaluating the technology at great scale and in an easy and economical way. This requires defining technology evaluations focusing on their reusability.

In the last few years, evaluation and benchmarking efforts have mainly focused on some types of technologies and on some of their metrics, namely, the interoperability of ontology development tools, the precision and recall of ontology alignment tools, and the efficiency and scalability of ontology repositories. But now we think that new efforts are required, first, to involve other Semantic Web technologies (ontology learning tools, ontology annotation tools, ontology population tools, etc.) and, second, to broaden the scope of these evaluations by considering a wider range of evaluation metrics for the technology (latency, robustness, security, usability, etc.).

The role of the research community when defining and performing benchmarking activities is crucial. Community-driven benchmarking connects experts and allows obtaining high quality results and increasing the credibility of the benchmarking and its results.

However, future research must focus on performing evaluations centred on the user of the Semantic Web technology. And it would be advisable to consider audiences from beyond the research community itself as recipients of the evaluation results.

re Ferences

Abran, A., Moore, J. W., Bourque, P., & Dupuis, R. (Ed.). (2004). *SWEBOK: Guide to the software engineering body of knowledge*. IEEE Press.

Basili, V. R., & Selby, R. W. (1991). Paradigms for experimentation and empirical studies in software engineering. *Reliability Engineering and System Safety*, *32*, 171-191.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Transactions on Software Engineering*, *12*(7), 733-743.

Basili, V. R. (1985, September). Quantitative evaluation of software methodology. In *1st Pan-Pacific Computer Conference*, Melbourne, Australia.

Bull, J. M., Smith, L. A., Westhead, M. D., Henty, D. S., & Davey, R. A. (1999). A methodology for benchmarking Java grande applications. In the *ACM 1999 conference on Java Grande* (pp. 81-88).

Camp, R. (1989). *Benchmarking: The search for industry best practices that lead to superior performance*. Milwaukee, ASQC Quality Press.

Freimut, B., Punter, T., Biffl, S., & Ciolkowski, M. (2002). *State-of-the art in empirical studies*. Technical Report ViSEK/007/E, Visek.

García-Castro, R., & Gómez-Pérez, A. (2005, November). Guidelines for benchmarking the performance of ontology management APIs. In Y. Gil, E. Motta, R. Benjamins, & M. Musen (Ed.), *4th International Semantic Web Conference (ISWC2005)*, *3729* in LNCS, 277-292. Galway, Ireland: Springer-Verlag.

García-Castro, R., Maynard, D., Wache, H., Foxvog, D., & González-Cabero, R. (2004). *D2.1.4 Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools*. Technical report, Knowledge Web.

García-Castro, R., Sure, Y., Zondler, M., Corby, O., Prieto-González, J., Paslaru Bontas, E., Nixon, L., & Mochol, M. (2006). *D1.2.2.1.1 Benchmarking the interoperability of ontology development tools using RDF(S) as interchange language*. Technical report, Knowledge Web.

Gediga, G., Hamborg, K., & Duntsch, I. (2002). Evaluation of software systems. In *Encyclopedia of Computer Science and Technology*, *44*, 166-192.

IEEE. (1998) *IEEE Std 1061-1998 IEEE Standard for a software quality metrics methodology.*

ISO/IEC (1999) *ISO/IEC 14598-1: Software product evaluation - Part 1: General overview.*

Juristo, N., & Moreno, A. (2001). *Basics of software engineering experimentation*. Kluwer Academic Publishers.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones P. W., Hoaglin, D. C., El-Emam, K., & Rosenberg, J. (2002). Preliminary guidelines for empirical research in Software Engineering. *IEEE Transactions on Software Engineering 28*(8), 721-734.

Kitchenham, B. (1996). *DESMET: A method for evaluating Software Engineering methods and tools*. Technical Report TR96-09, Department of Computer Science, University of Keele, Staffordshire, UK.

Park, R. E., Goethert, W. B., & Florac, W. A. (1996). *Goal-driven software measurement - a guidebook*. Technical Report CMU/SEI-96-HB-002, Software Engineering Institute.

Perry, D. E., Porter, A. A., & Votta L. G. (2000). Empirical studies of Software Engineering: a roadmap. In A. Finkelstein (Ed.), *The Future of Software Engineering,* 345-355. ACM Press.

Rakitin, S. R. (1997). *Software Verification and Validation, a practitioner's guide*. Artech House.

Shirazi, B., Welch, L. R., Ravindran, B., Cavanaugh, C., Yanamula, B., Brucks, R., & Huh, E. (1999). Dynbench: A dynamic benchmark suite for distributed real-time systems. In the *11th IPPS/SPDP'99 Workshops*,1335-1349. Springer-Verlag.

Sim, S., Easterbrook, S., & Holt, R. (2003). Using benchmarking to advance research: A challenge to software engineering. In the *25th International Conference on Software Engineering (ICSE'03)*,74-83. Portland, OR.

Spendolini, M. J. (1992). *The benchmarking book*. New York, NY: AMACOM.

Stefani, F., Macii, D., Moschitta, A., & Petri, D. (2003, June). FFT benchmarking for digital signal processing technologies. In the *17th IMEKO World Congress*. Dubrovnik, Croatia.

Sure, Y., & Corcho, O. (Ed.) (2003). Proceedings of the *2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, *87* of CEUR-WS. Florida, USA.

Weiss, A. R. (2002). *Dhrystone benchmark: History, analysis, scores and recommendations*. White paper, EEMBC Certification Laboratories, LLC.

Wireman, T. (2003). *Benchmarking best practices in maintenance management*. Industrial Press.

Wohlin, C., Aurum, A., Petersson, H., Shull, F., & Ciolkowski, M. (2002, June). Software inspection benchmarking - a qualitative and quantitative comparative opportunity. In the *8th International Software Metrics Symposium,* 118-130.

Add It Ion Al re Ad Ing

Ahmed, P., & Rafiq, M. (1998). Integrated benchmarking: a holistic examination of select techniques for benchmarking analysis. *Benchmarking for Quality Management and Technology* 5, 225-242.

Basili, V. R., Caldiera, G., & Rombach, D. H. (1994). The Goal Question Metric approach. *Encyclopedia of Software Engineering*, 528-532. Wiley.

Basili, V. R. (1993). The experimental paradigm in Software Engineering: Critical assessment and future directions. In the *International Workshop on Experimental Software Engineering Issues*, 3-12. Springer-Verlag.

Beitz, A., & Wieczorek, I. (2000). Applying benchmarking to learn from best practices. *Product Focused Software Process Improvement, Second International Conference (PROFES 2000)*, 59-72.

Brickley, D., Guha, R. V. (Ed.) (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation 10 February 2004.

Brown, A., & Wallnau, K. (1996). A framework for evaluating software technology. *IEEE Software, 13*, 39-49.

Corcho, O. (2005). *A layered declarative approach to ontology translation with knowledge preservation*. Volume 116 of Frontiers in Artificial Intelligence and its Applications. IOS Press.

Dongarra, J., Martin, J. L., & Worlton, J. (1987). Computer benchmarking: paths and pitfalls. *IEEE Spectrum, 24*(7), 38-43.

Duineveld, A. J., Stoter, R., Weiden, M. R., Kenepa, B., & Benjamins V. R. (1999). Wondertools? A comparative study of ontological engineering tools. In the *12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*, Banff, Canada: Kluwer Academic Publishers.

Dujmovic, J. J., (1998). Evaluation and design of benchmark suites. Chapter 12 in *State-of-the-art*

in performance modeling and simulation: theory, techniques and tutorials,287-323. Gordon and Breach Publishers.

Feitelson, D. G. (2005). *Experimental computer science:The Need for a Cultural Change*.

Fenton, N. (1991). *Software metrics - a rigorous approach*. Chapman & Hall.

Fenton, N., & Neil, M. (2000). Software metrics: Roadmap. In the *Conference on the future of Software Engineering*, 357-370. ACM Press.

Fernandez, P., McCarthy, I., & Rakotobe-Joel, T. (2001). An evolutionary approach to benchmarking. *Benchmarking: An International Journal, 8*, 281-305.

García-Castro, R. (2006). Benchmarking como herramienta de transferencia tecnológica Invited talk in the *3er Encuentro Internacional de Investigadores en Informática*. Popayán, Colombia.

García-Castro, R. (2006). Keynote: Towards the improvement of the Semantic Web technology. In the *Second International Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2006).*

García-Castro, R., & Gómez-Pérez, A. (2006). Benchmark suites for improving the RDF(S) importers and exporters of ontology development tools. In the *3rd European Semantic Web Conference (ESWC2006)*, 155-169. LNCS-4011.

García-Castro, R., Gómez-Pérez, A. (2006). Interoperability of Protégé using RDF(S) as interchange language. In the *9th International Protégé Conference*.

García-Castro, R. (2006) Keynote: Tecnologías de la Web Semántica: cómo funcionan y cómo interoperan. In the *4th Seminario Internacional Tecnologías Internet*. Popayán, Colombia.

García-Castro, R., & Gómez-Pérez, A. (2005). A method for performing an exhaustive evaluation of RDF(S) importers. In the *Workshop on Scalable Semantic Web Knowledge Based Systems (SSWS2005).*

García-Castro, R. (2005). *D2.1.5 prototypes of tools and benchmark suites for benchmarking ontology building tools*. Technical report, Knowledge Web.

Gee, D., Jones, K., Kreitz, D., Nevell, S., O'Connor, B., & Ness, B. V. (2001). Using performance information to drive improvement. *Performance-Based Management Special Interest Group, 6*.

Goodman, P. (1993). *Practical implementation of software metrics*. McGraw Hill.

Grady, R., & Caswell, D. (1987). *Software metrics: Establishing a company-wide program*. Prentice-Hall.

Jones, C. (1995, October). Software benchmarking. *IEEE Computer*, 102-103.

Kitchenham, B., Linkman, S., & Law, D. (1994). Critical review of quantitative assessment. *Software Engineering Journal*, *9*, 43-53.

Kitchenham, B., Pfleeger, S., & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering, 21*, 929-944.

Kraft, J. (1997). *The Department of the Navy benchmarking handbook: a systems view*. Department of the Navy.

Lankford, W. (2000). Benchmarking: understanding the basics. *Coastal Business Journal*.

Lukowicz, P., Tichy, W. F., Prechelt, L., & Heinz E.A. (1995). Experimental evaluation in computer science: A quantitative study. *The Journal of Systems and Software*, 28(1),1-18.

Manola, F., & Miller, E. (2004, February 10). *RDF Primer.* W3C Recommendation.

OntoWeb (2002). *D1.3: A survey on ontology tools*. Technical report, IST OntoWeb Thematic Network.

Pfleeger, S. L. (1999). Understanding and improving technology transfer in software engineering. *Journal of Systems and Software* 47,111-124.

Sim, S. (2003). *A theory of benchmarking with applications to software reverse engineering*. PhD thesis. University of Toronto.

Sole, T., & Bist, G. (1995). Benchmarking in technical information. *IEEE Transactions on Professional Communication* 38, 77-82.

Tichy, W. (1998). Should computer scientists experiment more? *Computer* 31, 32-40.

Wache, H., Serafini, L., & García-Castro, R. (2004). *D2.1.1 survey of scalability techniques for reasoning with ontologies*. Technical report, KnowledgeWeb.

Wireman, T. (2003). *Benchmarking best practices in maintenance management*. Industrial Press.

endnotes

[a] http://knowledgeweb.semanticweb.org/

[b] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/

[c] http://kaon.semanticweb.org/

[d] http://protege.stanford.edu/

[e] http://webode.dia.fi.upm.es/

[f] http://www-sop.inria.fr/acacia/soft/corese/

[g] http://jena.sourceforge.net/

[h] http://www.openrdf.org/

[i] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/

[j] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/rdfs_import_benchmark_suite.html

[k] http://knowledgeweb.semanticweb.org/benchmarking_interoperability/rdfs/rdfs_export_benchmark_suite.html

[l] The names of the tools have been shortened in the heading of the table: KAON=K, Protégé=P and WebODE=W.

Appendix: Questions For discussion

beginner:

1. Which are the main characteristics of benchmarking?
2. Which is the goal of the *Recalibration* task in the benchmarking methodology?
3. Which are the factors that influence the correct interchange of an ontology between two Semantic Web tools?
4. When exporting one ontology from an ontology development tool to RDF(S) having interoperability in mind, is it advisable to export the complete definition of all its components?

Intermediate:

1. Which are the differences between evaluation and benchmarking?
2. Are the users of the software involved in the benchmarking?
3. Is management support needed in the *Improve* phase of the methodology?
4. Which RDF(S) components can be represented in KAON, Protégé and WebODE?

Advanced:

1. Which resources are needed for performing a benchmarking activity?
2. Why are there three different evaluation criteria to define the results of the RDF(S) Import, Export and Interoperability benchmark suites?
3. Why is it not enough to have a single ontology representation language to achieve interoperability between the Semantic Web technologies?

practical exercises:

1. Select one conference paper that presents some evaluation or benchmarking approach and then evaluate its reusability according to the desirable properties of a benchmark suite and the recommendations given for software evaluation in benchmarking activities.
2. Create a mid-size ontology using one ontology development tool. Can you anticipate the consequences of exporting that ontology to RDF(S)? And of importing it into another ontology development tool?
3. Then, export the ontology to RDF(S). Was your prediction correct? Has it had information addition or loss?
4. Finally, import the exported ontology into the other ontology development tool. Was your prediction correct? Has it had information addition or loss?

Answers to the Questions For Discussion

beginner:

1. The main characteristics of benchmarking are continuous improvement and the search for best practices.
2. The goal of the *Recalibration* task is to improve the benchmarking process by recalibrating it and applying the lessons learnt while performing it.
3. The factors that influence the correct interchange of an ontology between two tools are the combinations of components of the knowledge model of the ontology development tool and the naming of the components present in the ontology.
4. It is advisable to export the complete definition of all its components only for components commonly used by ontology development tools.

Intermediate:

1. Benchmarking is a continuous process, whereas an evaluation is a punctual activity. In addition, benchmarking involves evaluating software but its goals are to obtain a continuous improvement on the software and the practices used when developing the tools.
2. Yes, the users of the software are identified in the *Participant identification* task and in the *Findings communication* task.
3. Yes, it is needed to implement the necessary changes in the software and in the organisation processes affecting the software.
4. Classes, class hierarchies without cycles, datatype properties with a class as a domain and a string range, object properties with a domain and a range, instances of a single class, instances related by object properties, and instances related by datatype properties with a string range.

Advanced:

1. The resources needed are human resources though some equipment and travel resources are also required, and these are mainly used in three tasks: benchmarking organisation, experimentation definition and execution, and result analysis.
2. Because these three evaluation criteria are necessary to represent the different situations and behaviours that can occur when two tools interchange one ontology.
3. Because different types of users need different tools; existing tools have different knowledge representation models; and tools need to translate their ontologies from their knowledge models to the common ontology representation language.