How much semantic data on small devices?

Conference or Workshop Item

How much Semantic Data on Small Devices?*

Mathieu d'Aquin, Andriy Nikolov, Enrico Motta

Knowledge Media Institute, The Open University, Milton Keynes, UK
{m.daquin, a.nikolov, e.motta}@open.ac.uk

Abstract. Semantic tools such as triple stores, reasoners and query engines tend to be designed for large-scale applications. However, with the rise of sensor networks, smart-phones and smart-appliances, new scenarios appear where small devices with restricted resources have to handle limited amounts of data. It is therefore important to assess how existing semantic tools behave on such small devices, and how much data they can reasonably handle. There exist benchmarks for comparing triple stores and query engines, but these benchmarks are targeting large-scale applications and would not be applicable in the considered scenarios. In this paper, we describe a set of small to medium scale benchmarks explicitly targeting applications on small devices. We describe the result of applying these benchmarks on three different tools (Jena, Sesame and Mulgara) on the smallest existing netbook (the Asus EEE PC 700), showing how they can be used to test and compare semantic tools in resource-limited environments.

1 Introduction

With the rise of sensor networks, smart-phones and smart-appliances, new scenarios appear where a number of small devices with restricted resources each have to handle limited amounts of data. In particular, the SmartProducts project [1] is dedicated to the development of "smart products" (namely, cars, airplanes and kitchen appliances) which embed "proactive knowledge" to help customers, designers, and workers in communicating and collaborating with them. Concretely, Smart Products rely on a platform using small computing devices (such as gumstix[1]) which process, exploit, and expose knowledge related to the product they are attached to through the use of semantic technologies.

While building such a platform, an obvious issue concerns the performance of semantic technologies on the considered hardware. Indeed, existing tools are usually designed for large scale applications and data, deployed on high performance servers, and possibly taking benefit from distributed computing approaches. A few initiatives have emerged that aim at providing tools dedicated to small devices[2], but these are not yet mature enough to be employed in the considered

* Part of this research has been funded under the EC 7th Framework Programme in the context of the SmartProducts project (231204).

[1] http://www.gumstix.com/

[2] see e.g., MobileRDF (http://www.hedenus.de/rdf/) and microJena (http://poseidon.elet.polimi.it/ca/?page_id=59)

scenarios. On the other hand, it is unclear whether popular systems such as Jena[3] or Sesame[4] could actually be used, how they would perform, and how they would compare on resource-limited hardware. Of course, it can always be argued that semantic processing can be applied remotely, on a server, so that the small device only has to act as an interface to semantic data, without having to handle it directly. In such cases however, mechanisms have to be put in place to ensure the availability of this data in all the different environments in which the device might be used, while maintaining appropriate levels of security and privacy. For this reason, one of the main rational underlying this work is to gain the ability to answer the question, "How much semantic data can we store and process on small devices?", so that we can also evaluate, in a given application, how much of the data *have to* be processes externally.

Several benchmarks have been devised to assess and compare the performance of semantic tools (see e.g., [2, 3]). However, here again, the current focus on large-scale applications makes these benchmarks inadequate to answer the above question. They indeed assume the availability of sufficient resources to run the considered tools on the large amounts of test data they contain. In addition, as they work at large scale, these benchmarks tend to focus only on two criteria to evaluate performance: the size of the data and the response time. When working at a smaller scale, other characteristics than size (e.g., the distribution of entities in classes, properties and individuals) can have a significant impact on the tools' performance. More importantly, to test the ability of a particular tool to run on a small device, other performance criteria need to be considered, which are often assumed to be available in sufficient quantities in large-scale benchmarks (namely, memory and disk space).

For these reasons, we created a set of "smaller-scale" benchmarks for tools implementing semantic technologies. In practice, each of these benchmarks corresponds to a set of ontologies (i.e., semantic documents) varying in size and with common characteristics (in terms of complexity and distribution of entities). We also use a set of eight generic queries of varying complexities to be executed on each of the ontologies in each of the benchmarks. Therefore, running each benchmark individually allows us to analyze the behavior of the considered tools in specific situations (according to particular data characteristics) and comparing the results of different benchmarks helps in understanding the impact of parameters other than the size of the ontology on various performance measures.

We experimented with running the created benchmarks on three popular tools—Jena, Sesame and Mulgara—with different configurations, on a very limited netbook (the Asus EEE PC 700, also called 2G Surf). This netbook has specifications in similar ranges to the kind of devices that are envisaged in the SmartProducts project. Hence, measuring response time, memory consumption and disk space while running our benchmarks on this device allows us to evaluate the ability and limitations of each of the tested tools if employed in concrete, small-scale scenarios.

[3] http://jena.sourceforge.net/
[4] http://openrdf.org

2 Benchmarks for small to medium scale semantic applications

In this section, we propose a new set of benchmarks for small to medium scale applications. Each benchmark is made of two components: 1- a set of ontologies (semantic documents) of varying sizes; and 2- a set of queries of varying complexities (common to all the benchmarks).

2.1 Ontology Sets

The sets of ontologies to be used for testing semantic tools have been built following two main requirements. First and most obviously, they had to cover ontology sizes from very small (just a few triples) to medium-scale ones (hundreds of thousands of triples). As expected, and shown in the next section, medium-scale ontologies represent the limit of what the best performing tools can handle on small devices. Second, these sets of ontologies should take into account the fact that, especially at small-scale, size is not the only parameter that might affect the performance of semantic tools. It is therefore important that, within each set, the ontologies vary in size, but stay relatively homogeneous with respect to these other characteristics. In this way, each set can be used to test the behavior of semantic tools on a particular type of ontologies (e.g., simple ontologies containing a large number of individuals), while comparing the results obtained with different ontology sets allows us to assess the impact of certain ontology characteristics on the performance of the considered tools. In addition, we also consider that relying on real-life ontologies (i.e., ontologies not automatically generated or composed for the purpose of the benchmark) would lead to more exploitable results.

In order to fulfill these requirements, we took advantage of the Watson Semantic Web search engine[5] to retrieve sets of real life ontologies[6] of small to medium sizes. We first devised a script to build sets of ontologies from Watson grouping together ontologies having similar characteristics, therefore building homogenous sets with respect to these characteristics. The parameters employed for building these groups are the ratio $\frac{Number\ of\ Properties}{Number\ of\ Classes}$, the ratio $\frac{Number\ of\ Individuals}{Number\ of\ Classes}$ and the complexity of the ontological description, as expressed by the underlying description logic (e.g., $\mathcal{ALCH}$). The 2 first parameters were allowed a derivation of more or less 50% from the average in the group (ontologies rarely have exactly the same values for these characteristics). As a result of this automatic process, we obtained 99 different sets of ontologies. We then manually selected amongst these sets the ones to be used for our benchmarks, considering only the sets containing appropriate ranges of sizes (see summary of the selected benchmark ontology sets in Table 1).

[5] http://watson.kmi.open.ac.uk

[6] Here we use to word ontology to refer to any semantic document containing RDF descriptions, including documents containing individuals.

Table 1. Summary of the 10 benchmark ontology sets. The name of each set is given according to its number in the original automatic process. Size is in number of triples.

Name	Ontos	Size range	Ratio prop./class	Ratio ind./class	DL Expressivity
12	9	9-2742	0.65-1.0	1.0-2.0	$\mathcal{ALO}$
37	7	27-3688	0.21-0.48	0.07-0.14	$\mathcal{ALH}$
39	79	2-8502	no class	no class	-
43	56	17-3696	0.66-2.0	4.5-20.5	-
53	21	3208-658808	no property	no individual	$\mathcal{EL}$
54	11	1514-153298	no property	no individual	$\mathcal{ELR}+$
56	20	8-3657	no class	no class	-
58	35	7-4959	1.41-4.0	no individual	$\mathcal{AL}$
66	17	1-2759	no property	no class	-
93	11	43-5132	1.0-2.0	13.0-22.09	-

2.2 Queries

Since our benchmarks are derived from various, automatically selected ontologies, we needed a set of queries generic enough to give results on most of these datasets. We therefore devised eight queries of varying complexities, which are based on the vocabularies of the RDFS, OWL, and DAML+OIL languages:

1. *Select all labels*. This is a basic query which returns all *rdfs:label* datatype values.
2. *Select all comments*. This query returns all *rdfs:comment* values. Usually these values are longer than *rdfs:label*, but more scarce.
3. *Select all labels and comments*. This query checks the ability of the tool to deal with OPTIONAL clauses.
4. *Select all RDFS classes*. This is a basic query which returns RDF resources of type *rdfs:Class*. Its results can be different depending on whether reasoning is applied and whether the tool is aware of the *subClassOf* relation between RDFS and OWL classes.
5. *Select all classes*. This query explicitly searches also for OWL and DAML classes in addition to RDFS classes. The query constructs a union of several clauses.
6. *Select all instances of all classes*. This query contains the clauses of the previous query augmented with joint patterns. The set of results for this query varies depending on whether reasoning is applied: reasoning produces more answers to the query because each instance belonging to a subclass is inferred to belong to a superclass as well.
7. *Select all properties applied to instances of all classes*. This query adds an additional joint pattern to the previous one.
8. *Select all properties by their domain*. This query centers on properties and is also supposed to return different sets of results depending on the reasoning mechanism enabled.

The SPARQL queries corresponding to the descriptions above are also available at http://watson.kmi.open.ac.uk/small-scale-benchmarks.

3 Applying the benchmarks

In our experiments, we focused on two goals: 1- Testing the behaviour of popular semantic data storage tools on a resource-constrained device; and 2- comparing the performance ranking of the same tools when processing small and to medium scale datasets.

As a hardware platform to conduct the experiments we selected the Asus EEE PC 2G Surf netbook (the oldest version of Asus EEE netbooks) with 900 MHz CPU, 512 MB RAM and Puppy Linux[7] installed. This provides a good reference platform as it has similar specifications to common embedded computing devices[8], as well as to top-of-the-range smart-phones[9] while being reasonably convenient to use.

We have selected three of the popular semantic data store tools, which provide Java API interface: Jena, Sesame 2, and Mulgara[10]. We did not test some other storage tools specifically designed for handling large scale datasets such as Virtuoso or 4store because of their high initial resource requirements (e.g., Virtuoso installation requires 62MB of disk space). Because Jena and Sesame provide ontological reasoning capabilities, we tested these tools in two modes: without reasoning and with RDFS reasoning to estimate the additional resource usage caused by inferencing. The following test system configurations were used:

- Jena TDB v0.8.2.
- Jena TDB v0.8.2 with the Jena native RDFS reasoner.
- Sesame v2.2.4 with the RDF Native SAIL repository.
- Sesame v2.2.4 with the RDFS Native SAIL repository (i.e., with reasoning).
- Mulgara v2.1.6 (Lite).

Each test consisted of the following steps:

1. Creating an empty data store.
2. Loading an ontology into the data store.
3. Running queries 1-8.
4. Measuring the disk space taken by the data store.
5. Cleaning the data store (physically deleting the corresponding files).

For each step we measured the time cost and the size of the Java heap space used by the virtual machine. Because of the resource limitations of the chosen netbook, we have restricted the maximum heap size to 400 MB.

3.1 Results

Table 2 gives a summary of all the results obtained after running the 5 different tool configurations described above on our 10 benchmark ontology sets. We can distinguish 2 sets of measures in this table: measures related to the performance

[7] http://www.puppylinux.com/

[8] For example, the SmartProduct project employs Overo Air gumstix with 600 MHz CPU and 256 MB RAM.

[9] For example, the Apple iPhone 3GS has a 600Mhz CPU and 256 MB of RAM, and the Sony Ericsson XPeria X10 has 1GHz CPU and 1GB of memory.

[10] http://www.mulgara.org/

Table 2. Average measures for the different tools using the different benchmark sets.

Benchmark	12	37	39	43	53	54	56	58	66	93
Jena No Reasoning										
# of ontologies treated	9	7	78	56	19	11	20	35	17	11
% of overall size processed	100%	100%	80%	100%	43%	100%	100%	100%	100%	100%
Avg. load time/triple (ms)	54	26	48	12	1	2	37	32	271	13
Avg. memory/triple (KB)	267	86	265	80	1	1	180	199	1023	32
Avg. disk space/triple (KB)	9	2	14	3	0.17	0.19	4	5	27	0.93
Avg. time query/Ktriple (ms)	2460	969	1722	604	232	149	1448	1155	8067	442
Avg. # of query results	22	114	0.87	13	3318	2874	7	12	0.97	80
Avg. time/K query result (ms)	6089	4564	601	1297	335272	147	6456	1480	1998	2913
Jena RDFS Reasoning										
# of ontologies treated	9	6	78	56	2	1	20	34	17	11
% of overall size processed	100%	39%	80%	100%	0%	0%	100%	44%	100%	100%
Avg. load time/triple (ms)	65	32	50	12	2	6	38	33	254	13
Avg. memory/triple (KB)	284	106	282	85	2	5	191	216	1035	34
Avg. disk space/triple (KB)	7	2	14	3	0.17	0.19	4	5	27	0.93
Avg. time query/Ktriple (ms)	13317	10363	10717	4110	15078	28307	10732	37353	44078	2710
Avg. # of query results	75	113	240	357	551	297	99	130	87	216
Avg. time/K query result (ms)	17403	64531	5606	3144	955122	1381216	13461	79262	23116	6481
Sesame No Reasoning										
# of ontologies treated	9	7	78	56	21	11	14	7	17	11
% of overall size processed	100%	100%	80%	100%	100%	100%	9%	1%	100%	100%
Avg. load time/triple (ms)	12	6	7	3	1	1	9	17	38	3
Avg. memory/triple (KB)	32	12	29	10	0.16	0.23	28	62	115	5
Avg. disk space/triple (KB)	0.51	0.24	0.53	0.21	0.1	0.12	0.47	1	1	0.15
Avg. time query/Ktriple (ms)	2334	987	1235	336	43	60	1738	3955	9816	408
Avg. # of query results	22	114	0.87	53	8329	2874	2	1	1	80
Avg. time/K query result (ms)	3899	2905	710	1877	117	272	1984	4587	750	1844
Sesame RDFS Reasoning										
# of ontologies treated	9	7	78	56	21	11	14	7	17	11
% of overall size processed	100%	100%	80%	100%	100%	100%	9%	1%	100%	100%
Avg. load time/triple (ms)	16	10	8	4	3	4	11	25	50	5
Avg. memory/triple (KB)	55	17	50	17	0.23	0.34	46	107	160	8
Avg. disk space/triple (KB)	1	0.45	0.96	0.4	0.14	0.18	0.96	2	3	0.25
Avg. time query/Ktriple (ms)	2942	1156	1578	401	122	152	2100	5670	14277	511
Avg. # of query results	58	153	29	102	10617	3882	22	16	53	154
Avg. time/K query result (ms)	3389	3172	1272	610	99367	31836	1405	2685	1585	1322
Mulgara										
# of ontologies treated	9	7	78	56	19	11	20	35	17	11
% of overall size processed	100%	100%	80%	100%	43%	100%	100%	100%	100%	100%
Avg. load time/triple (ms)	141	54	132	49	2	2	105	110	595	27
Avg. memory/triple (KB)	139	49	141	41	0.43	0.62	96	101	522	17
Avg. disk space/triple (KB)	3778	1650	6090	2175	32	43	5757	7261	43412	1451
Avg. time query/Ktriple (ms)	9909	5875	5340	2651	1291	1399	4896	4719	27589	2704
Avg. # of query results	22	114	0.87	53	3318	2874	7	12	0.97	80
Avg. time/K query result (ms)	14916	9056	2942	16256	885	1390	3780	6655	4713	5466

of the tools (e.g., loading time) and measures related to their robustness (i.e., their ability to process large numbers of heterogeneous, real-life ontologies). Indeed, concerning robustness, each benchmark was run automatically for each tool, processing the ontologies in the corresponding set in an order of size. When a tool crashed on a particular ontology, unless an obvious error could be corrected, the test on the corresponding benchmark was interrupted. As can be seen from Table 2, the different tools tend to break on different benchmarks (with benchmark 39 being commonly problematic to all of them). In addition, while the application of RDFS inferences does not seem to affect the ability of Sesame to process ontologies (it consistently breaks on the same ontologies, and no other), Jena tends to be a lot less robust when reasoning is applied, especially on medium size ontologies, most likely due to the increase in its need for resources. All together, Mulgara and Jena without reasoning appear to be the most robust tools with 8 out of 10 benchmarks being covered at 100% (here, percentage is expressed with respect to the sum of all the triples in all the ontologies of the benchmark). The fact that they can process exactly the same number of ontologies can be explained by the fact that Mulgara uses Jena as a parser.

Regarding performance measures, we give more details below on the behavior of each tool configuration concerning loading time, memory consumption, disk space and query response time. However, generally, it appears quite clearly that Sesame (without reasoning) tends to outperform the other tools at small-scale. As we will see, this is less true at medium-scale, and other benchmarks have shown that, at large-scale, the difference between Sesame and Jena tends to be inverted [3]. This can be explained by the fact that Jena and Mulgara generally allocate larger amounts of resources straight from the start, while an "empty Sesame" is very lightweight. In other terms, it seems that the "fixed cost" associated with processing ontologies with Sesame is significantly lower than the one of Jena and Mulgara, whereas the "variable cost" appears to be higher. While this turns out to be a disadvantage at large-scale, it certainly makes Sesame a stronger candidate in the scenarios we consider here, i.e., small-scale, resource-limited applications.

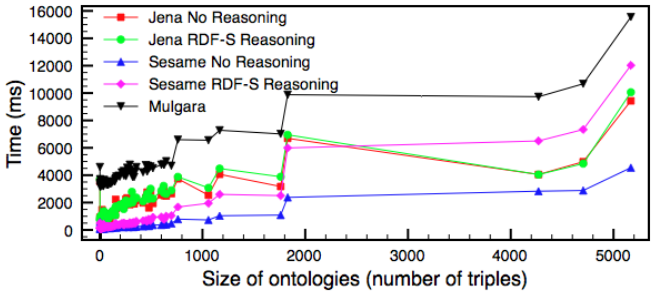

Fig. 1. Loading time with respect to size of the ontologies in benchmark 39.

Loading time refers to the time (expressed in milliseconds) taken by each tool to parse the file containing the ontology, create the internal data structures necessary to process this ontology, and store the information contained in the ontology in these internal structures. Naturally, this measure depends a lot on

the storage device employed, but the way each tool represents the ontologies internally also has a significant impact. Indeed, as can be seen from Table 2, loading ontologies takes significantly less time for Sesame than for any other of the tested tool. Mulgara, on the other hand, creates many indexes for the data, which clearly impacts negatively on this measure. It can be noticed that applying RDFS reasoning with Sesame influences loading time, with supposedly some inferences being drawn already when initializing the ontology model, while it is not the case for Jena. Figure 1 shows a typical evolution of loading time with respect to the size of the ontology (using benchmark 39). Here we can see that Sesame was able to load more than 5000 triples in less than 5 seconds without reasoning, compared to around 12 seconds with reasoning. Mulgara took almost 16 seconds. It is interesting to see also how Jena loads ontologies apparently independently from the use of RDFS reasoning.

Having compared the results obtained for different benchmarks, we can observe that the tools behave consistently with respect to loading time independently from variables other than size. Sesame with reasoning, however, performs better than Mulgara for small ontologies only, and ends up taking longer for ontologies above a few thousand triples. Also, loading time appears to be slightly higher for ontologies with the same number of triples but more expressive description logics. This could be explained by a higher density of description for classes in these cases.

Memory consumption is described in Table 2 by providing the average memory space required (in kilo-bytes) per triple in each ontology. As for loading time, this is measured right after having loaded the ontology from the file, evaluating the amount of memory required to make the model accessible. Here, Sesame appears again to be the best performing tool at small scale, both when applying reasoning and not. While using reasoning in Sesame add an increment to memory consumption, for Jena reasoning does not have a significant impact on the memory consumption at loading time. Regarding memory consumption, Mulgara seems to be generally less demanding than Jena (both with reasoning and without). While differences appear on average between benchmarks of different sizes, within benchmarks, no correlation is visible between the size of the ontologies and the memory consumption (due to the use of persistent storage on disk), except for Jena which shows a slight, linear increase of memory consumption with size.

Disk space is also measured at loading time, calculating the overall size at the location on the local disk where each of the tools keep their stores. Here again, Sesame stands out as being the least demanding when reasoning is not applied, but also, to a smaller extent when reasoning is applied. Since Jena does not store the results of inferences, there are very little differences in terms of required disk space when applying RDFS reasoning and when not. While Jena and Sesame perform comparably, Mulgara can be seen as being extremely demanding in terms of disk space. Indeed, contrary to Sesame, which almost does not require any disk space at all when running without any ontology loaded, Mulgara allocates a lot of space for storage structures such as indexes at starting time (in our

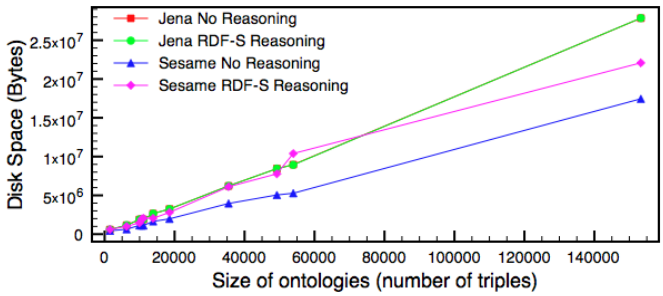

Fig. 2. Disk space consumption at loading time with respect to size of the ontologies in benchmark 54.

experiments, around 150MB), which makes it a very weak candidate in scenarios where small-scale ontologies are processed on devices with limited storage space. As shown in the graph Figure 2 looking at benchmark 54, disk space seems to increase linearly with the size of the ontology for Jena and Sesame. Mulgara is absent from the graph (as it was orders of magnitude higher than the others), but here as well, disk space consumption increases linearly, with a slightly lower factor than Sesame and Jena. In the example Figure 2, for more that 15 000 triples, Sesame takes between 15 MB and 20 MB of disk space depending on whether reasoning is applied, and Jena takes almost 30 MB.

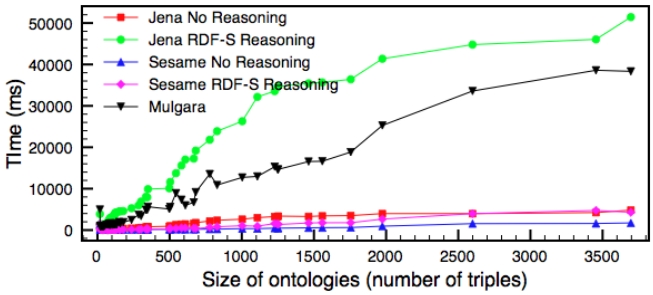

Fig. 3. Query response time (sum on all the 8 queries) with respect to size of the ontologies in benchmark 43.

Query time is measured when executing the 8 queries defined in Section 2.2. While analyzing the results for each individual query could help identifying some fine-grained behavior, we focus here on the global performance of the tools and consider the overall time to execute the 8 queries. Looking at the average results presented in Table 2, Sesame and Jena appear to be performing well, in similar ranges, when reasoning is not applied. However, as already noticed before, these two tools apply different strategies concerning inferencing. Indeed, Sesame applies and stores results of inferences when loading the ontology (making loading time higher), while Jena applies reasoning at query time. This difference appears very clearly in the results concerning query times. In accordance with its "pre-inferencing" strategy, Sesame provides results with inferences in times very close to the ones without inferencing. For Jena, however, applying reasoning leads to a

very significant increase in query time. This difference in behavior is visible in the graph Figure 3, showing the sum of the times required to execute our 8 queries in relation to the size of the ontologies in benchmark 43. As can be seen, even if it does not include any reasoning facility, Mulgara performs relatively badly compared to Jena and Sesame. Indeed, while Jena without reasoning, as well as Sesame both with and without reasoning are able to execute queries in near real-time even on such a small device as our netbook, Mulgara and Jena with reasoning would need up to 38 and 50 seconds respectively to provide results for an ontology of less than 4 000 triples.

Another interesting observation concerns the differences in the results of the queries. Indeed, looking at Table 2, the three tool configurations which do not apply reasoning obtain reasonably consistent results on our set of queries (on the benchmarks where they all managed to process the same number of ontologies), meaning that they interpret SPARQL queries in very similar ways. However, while analyzing the number of results obtained by the two tools applying reasoning, significant differences appear. The explanation to this phenomenon is that, in addition to having different strategies on when to apply reasoning, these tools implement RDFS inferences differently, with Jena generally obtaining more results (i.e., entailing more new statements). Additional investigations would be required to find out whether this is the result of different interpretations of the semantics of RDFS, of incomplete results from Sesame, or of incorrect results from Jena.

4 Conclusion

In this paper, we have established a set of small to medium scale benchmarks to test the performance of semantic tools on small devices with limited resources. Using these benchmarks and through extensive tests we have shown that tools such as Sesame, and to a smaller extent Jena, were able to cope reasonably well with small-scale ontologies on a very resource limited netbook. These results provided new insights into the behaviour of semantic data management tools in comparison with large-scale benchmarking tests of the same tools. Of course, the benchmarks we developed can be used to test any other semantic tool on any other platform, providing the availability of the necessary underlying infrastructure on such a platform. While the results obtained are encouraging, they also validate our intuition that existing semantic technologies are developed with large-scale applications in mind and that more work is needed to develop semantic software infrastructures dedicated to small-scale applications running with limited hardware resources.

References

1. Sabou, M., Kantorovitch, J., Nikolov, A., Tokmakoff, A., Zhou, X., Motta, E.: Position paper on realizing smart products: Challenges for semantic web technologies. In: Workshop: Semantic Sensor Networks (SSN09), ISWC. (2009)
2. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics **3**(2-3) (2005) 158–182
3. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International Journal on Semantic Web and Information Systems **5**(2) (2009) 1–24