Efficient Ontology-Based Modeling of Context-Aware In-Car Infotainment Systems

Benchmark Infrastructure and Design Guidelines

Daniel Lüddecke[1], Christoph Seidl[2], and Ina Schaefer[2]

[1] Volkswagen AG, Group Research, 38440 Wolfsburg, Germany,
daniel.lueddecke@volkswagen.de
[2] Technische Universität Braunschweig, 38106 Braunschweig, Germany,
{c.seidl, i.schaefer}@tu-braunschweig.de

Abstract. Context-aware systems, such as in-car infotainment systems, aim at improving the interaction between computers and humans by using contextual information about the system, the user and the environment. Previous work has shown that ontologies have significant benefits for modeling such context-aware systems, e.g., when inferring knowledge. However, the potential performance impact on the overall system when adopting ontologies, especially with rules, to model context-awareness is yet unknown. In this paper, we introduce a benchmark infrastructure and perform benchmarks on multiple ontologies of context-aware systems in order to determine factors that influence performance. From the results of these benchmarks, we derive guidelines for designing ontologies with rules for context-aware systems. These guidelines allow making conscious decisions about performance trade-offs and, in consequence, may improve suitability of ontologies for use in implementing industrial context-aware systems as they guide the creation of high-performance ontology-based context-aware systems.

1 Introduction

Context-aware systems aim at improving the interaction between computers and humans by using contextual information about the system, the user and their environment to provide suitable functionality for a particular context. For example, context-aware in-car infotainment systems may adapt to the respective drivers, their current situation and their intentions. In such a system, the car may, e.g., play rock music when the driver was identified as being tired. To realize such change in functionality, contextual values have to be captured and their effect on the system has to be specified in suitable models. In previous work [1], we showed that using ontologies to model context-aware systems has beneficial qualities, such as logical reasoning (e.g., *Pellet Reasoner*[1]), standardized APIs (e.g., *OWL API for Java*[2]), and extensive tool support (e.g., *Protégé*[3]).

[1] http://www.clarkparsia.com/pellet
[2] http://owlapi.sourceforge.net
[3] http://protege.stanford.edu

However, depending on the modeling of the context-aware system within the ontology, using this technology may result in a performance impact on the overall system, which may be hinder adoption within in-car infotainment systems as they have an inherent requirement on high performance due to frequent reasoning on the ontology. In this paper, we address this problem by introducing a benchmark infrastructure for ontology-based context-aware systems and by performing benchmarks on various differently modeled ontologies. From the results of these benchmarks, we derive design guidelines that may be used to model high-performance context-aware systems using ontologies.

This paper is structured as follows: In Section 2, we provide background information on context-aware systems and their modeling. In Section 3, we introduce the benchmark setup used to inspect the performance of ontology-based context-aware in-car infotainment systems. In Section 4, we describe the execution and present the results of our benchmarks by revealing significant influences certain properties of ontologies have on the performance of the overall system. In Section 5, we discuss these results and use them to derive design guidelines for building high-performance ontology-based context-aware systems. In Section 6, we elaborate on related work. Finally, in Section 7, we summarize the contributions and provide an outlook to future work to conclude the paper.

2 Background

Context-aware systems consider contextual values to alter their functionality to be suitable for a specific operator in a specific situation and a particular environment. Context-aware systems receive low level contextual information (e.g., from sensors), process these to high level contextual information and react to the change in context by altering their functionality appropriately [2]. Contextual information includes all values that may have an impact on the change of system behavior.

Context-aware systems from the automotive domain, especially in in-car infotainment systems, distinguish three main categories of contextual information: the driver, the car and their environment [3,4], e.g., for information such as the current stress level of the driver, available fuel amount of the car and the geographical position within the environment, respectively. To utilize contextual information in computer systems, the respective context values have to be captured in a specific model so that they may be processed further. A suitable notation for modeling context-aware systems are *ontologies* [2,5]. In our previous work, we exploited benefits to model context-aware in-car infotainment systems using ontologies [1] by using the Web Ontology Language (OWL) [6] with the Semantic Web Rule Language (SWRL) [7] as a rule-based extension to OWL. This approach allows the development of in-car infotainment systems that are able to behave differently with respect to the current situation of the car, the driver and their environment. The general idea is to use ontologies to model real world facts as classes of this ontology and relations between those classes. During run-time, observed data is added as individuals of those classes

to the ontology and reasoning techniques are used to infer knowledge that was not modeled explicitly during modeling-time. In our industrial practices, we observed that ontologies mostly vary in two facts: *a*) the amount of classes used to model the real world, and *b*) the use or disuse of SWRL rules. As the ontology contains all information about input and output data that enters or leaves the context-aware in-car infotainment system, it can be used to generate various code fragments that put data into the ontology or receive data from the ontology during run-time. However, capitalizing on these benefits by using ontologies may entail an impact on performance of the overall system due to the complexity of the necessary computations for reasoning depending on the concrete modeling of the context within the ontology. Although it may not be necessary for an in-car infotainment system to react in a range of milliseconds to a changing context, intense and time-consuming reasoning processes should be avoided to not block computational resources for other system tasks, which makes the processing of ontologies performance-critical. To create a suitable design of a context-aware system using ontologies, a conscious decision about the trade-off between the aforementioned benefits and the potential impact on performance is necessary.

3 Benchmark Setup

Despite the benefits of modeling context-aware systems with ontologies [1], there may be a performance impact on the overall system depending on how the system is modeled within the ontology. However, at the current state, no data exists on potential performance trade-offs entailed by employing ontologies for modeling context-aware systems, such as in-car infotainment systems. To remedy this shortcoming, we devised and performed a benchmark to inspect the performance impact entailed by various styles of modeling a context-aware system as an ontology.

In this section, we introduce the software setup that allows benchmarking the performance of ontology-based context-aware systems. Furthermore, we elaborate on the procedure to generate random ontologies to diversify the input to the benchmark in order to determine the performance of ontology-based context-aware systems.

3.1 Software Architecture

As foundation of the benchmark, we employ a component based architecture. The respective components are used to represent the individual functions found in a context-aware system as well as to synthesize data creation and consumption in the form of mock-ups. As we are using ontologies to understand the current situation of the user as described in [1], these components need to be able to *a*) add observed data to an active ontology during run-time, *b*) to use an ontology's reasoning techniques to infer knowledge during run-time, and *c*) to distribute this knowledge within the entire system. Figure 1 is a schematic

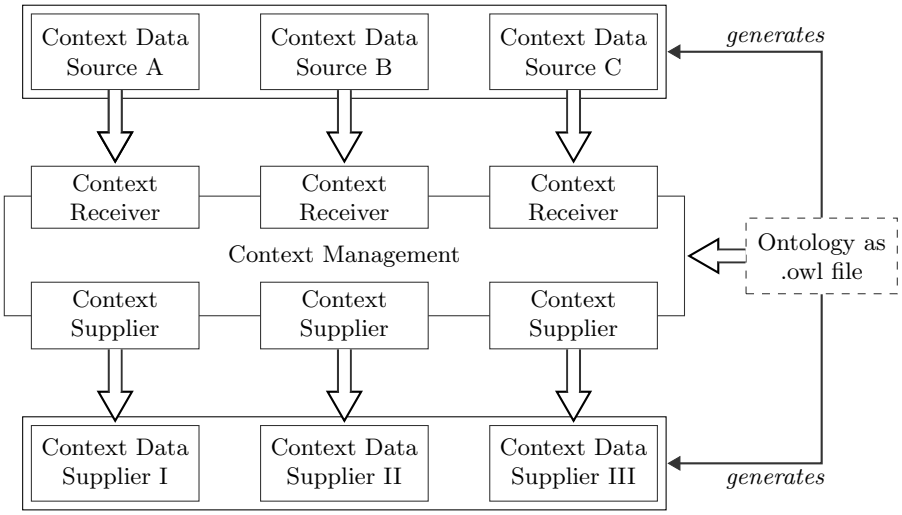


Fig. 1: Software components and data flow of the benchmark architecture.

overview of the software architecture used for executing benchmarks based on such software components.

Context Data Sources provide contextual information in a generic format for the *Context Management*. The *Context Management* loads the model in terms of an ontology saved in the OWL file format and creates individuals within the provided ontology for each contextual information sent by any *Context Data Source* and updates these individuals whenever it receives new data. Hence, the number of individuals will never exceed the number of classes within the ontology. Afterwards, the *Context Management* starts reasoning on the ontology to infer knowledge, e.g., the *Context Management* could reason about the driver's condition based on his or her interaction with the car. For this purpose, the benchmark infrastructure uses the *Pellet Reasoner*. In a last step, the *Context Management* queries the ontology for added inferred knowledge and notifies the corresponding *Context Data Supplier* of any changes.

As described in [1], a mapping between *Context Data Sources* and an ontology is used to ensure that sensor data can be added to the ontology during run-time. This mapping is done by annotating classes of the ontology. Classes that are annotated with the name of a *Context Data Sources* are called *Input Classes* and represent incoming data to the ontology. Classes that are annotated with the name of a *Context Data Supplier* are called *Output Classes* and represent description of the current situation, e.g., whether the driver is tired or awake.

When the *Context Management* launches, it creates generic *Context Receivers* that establish connections to every *Context Data Source*. In addition, names of *Context Data Suppliers* are attached to every *Output Class* defined within the ontology. Hence, the *Context Management* can again create generic *Context Sup-*

pliers that establish connections to their corresponding *Context Data Supplier* during run-time.

3.2 Input Data

It is our intent to benchmark a wide variety of ontologies for context-aware systems following different design approaches. Furthermore, to reliably determine the impact of different design approaches on performance, a significant size of an ontology is required, which exceeds the size of the case studies used in our previous work [1].

For these reasons, we decided to devise a generation algorithm for ontologies following different design approaches that have may be varied in size. Furthermore, we also created a procedure to generate appropriate software components that are used to interact with the ontology during run-time, e.g., *Context Data Sources* and *Context Data Supplier*. Our ontology generation algorithm is aligned with best practices of ontologies that were created by domain experts of ontology-based context-aware in-car infotainment systems during several industrial projects.

Ontology Generation To ensure that the results measured from our benchmark are not specific to a particular design approach for ontologies, we decided to generate several ontologies following different design approaches. For example, we align the depth of the inheritance hierarchy with our experience from industrial practice but allow parametrization of the number of direct subclasses. These are automatically generated with respect to a set of manually defined parameters:

- the amount of *Input* and *Output Classes*
- the amount of SWRL rules used for reasoning

Using these parameters, we are able to create a wide variety of ontologies that can be used as input to the benchmark as well as to generate *Context Data Sources* and *Context Data Supplier*. We assured that the generated ontologies closely resemble our industrial modeling practices for ontologies in a context-aware in-car infotainment.

Code Generation The basic implementation of the benchmark architecture presented in Figure 1 is provided as generic components, such as the *Context Management*, which can be reused in various scenarios. To further benchmark concrete context-aware systems based on ontologies, we generate *Context Data Sources* and *Context Data Supplier* from a specified ontology. During run-time, the generated *Context Data Sources* fire data change events for each of their *Input Classes* in arbitrary intervals to ensure a constant system load. *Context Data Suppliers* receive notifications whenever one of their *Output Classes* was inferred by the *Context Management*. Hence, the model and every software component of our benchmark are either generic or automatically generated by a corresponding code/ontology generator.

4 Execution and Results

Using the previously described setup, we performed benchmarks for various different ontologies. This section elaborates on the method of execution for the benchmarks and presents the respective results regarding the performance of using ontologies to model context-aware systems.

4.1 Execution

We make two assumptions how an ontology's size and the amount of SWRL rules used for reasoning may influence the performance of the overall system:

Assumption 1: An ontology's size has a negative effect on the performance of the overall system. Bigger ontologies are expected to slow down the system performance.

Assumption 2: The number of SWRL rules used for reasoning has a negative impact on the performance of the overall system. Ontologies with more SWRL rules are exected to slow down the system performance. However, to renounce on SWRL rules would massively limit expressiveness of our model of contextual information.

To check these assumptions, we create two metrics representing two different time frames during run-time of an ontology-based context-aware system:

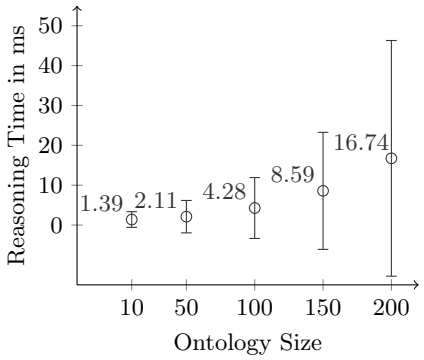
- Reasoning Time: The time it takes the Pellet Reasoner to do the reasoning on the ontology during run-time.
- Query Time: The time that is needed to query the ontology for inferred knowledge and to notify a *Context Data Supplier*.

4.2 Assumption 1

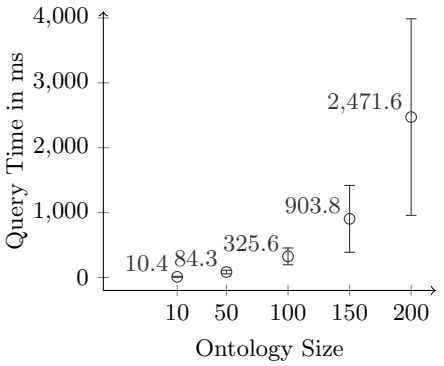
To check our first assumption, we generated multiple ontologies of different sizes (i.e., 10, 50, 100, 150 and 200 classes). For each size, we generated three ontologies. Every one of these was stressed in three separate runs by receiving data from three different *Context Data Sources*. We stopped the recording as soon as the reasoning had made $1,000$ cycles.

Figure 2a shows that the reasoning time of ontologies increases exponentially with an increasing size of the ontology. Even more important, the standard deviation of the reasoning time increases dramatically with size. The maximum reasoning time that occurred during our benchmark of ontologies with 200 input classes was as high as $2,128.0ms$ and the minimum was as low as $1.0ms$ by a mean of $16.74ms$. The standard deviation was at $29.56ms$. This implies, that the reasoning time of larger ontologies cannot be predicted as accurately as that of smaller ontologies.

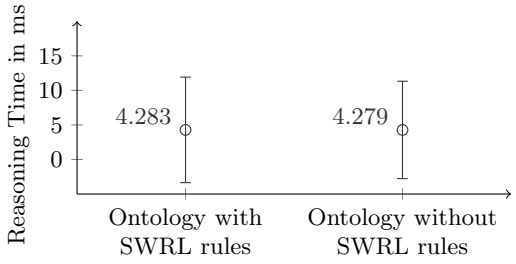
Figure 2b shows a significant increase of mean query time as well as its standard deviation. This means that it takes much more time to query larger



(a) Mean reasoning time and standard deviation compared to ontology size.



(b) Mean query time and standard deviation compared to ontology size.



(c) Mean reasoning time and standard deviation compared to ontologies with and without SWRL rules.

Fig. 2: Reasoning and query time compared to ontology size and usage of SWRL rules.

ontologies for inferred knowledge than it takes for smaller ontologies. In addition, the accuracy of predictions regarding the required query time deteriorates with the size of an ontology as can be seen by the high standard deviation. For ontologies with 200 classes, we measured query times between $20ms$ and $12,765ms$ at a mean of $2472ms$ and a standard deviation of $1514ms$.

4.3 Assumption 2

To check our second assumption, we compared the execution time of the previously used ontologies with that of similar ontologies without rules. For this purpose, we removed all SWRL rules from all three ontologies of size 100 and again performed $1,000$ reasoning cycles in three separate runs.

Figure 2c compares the results of ontologies with and without SWRL rules. The results show that the existence of SWRL rules does not have any significant

effect on the reasoning time of an ontology. The mean reasoning time is almost unaffected ($4.283ms$ with SWRL rules compared to $4.279ms$ without SWRL rules) and standard deviation increases only slightly when using SWRL rules ($7.636ms$ with SWRL rules compared to $7.044ms$ without SWRL rules). We also recorded a higher maximum value when reasoning ontologies with SWRL rules ($531ms$ with SWRL rules compared to $369ms$ without SWRL rules).

5 Discussion and Design Guidelines

In this section, we discuss the implications of the results of our benchmarks presented in Section 4. Furthermore, we use the insights gained from these results to derive guidelines for the modeling of high-performance ontology-based context-aware in-car infotainment systems.

For one, the benchmark results show that the reasoning time increases significantly with the number of classes in the ontology, which may lead to a bad performance of the overall system as the reasoner blocks computational resources during reasoning. Hence, our first assumption can be confirmed. Furthermore, the results also show that SWRL rules within ontologies do not have any significant impact on the overall system performance as mean and standard deviation are very close with and without SWRL rules. Hence, our second assumption cannot be confirmed. This provides liberty to modelers creating an ontology-based context-aware in-car infotainment system with regard to using SWRL rules in their ontologies and, hence, build even more powerful ontologies.

Derived from the performed benchmarks and presented results, we define the following design guidelines for modeling high-performance ontology-based context-aware in-car infotainment systems.

Guideline 1: Reconsider an ontology's number of classes.
As our benchmark results have shown, an increasing number of classes within an ontology has a negative impact on the performance of an ontology-based context-aware in-car infotainment system. In absolute terms, we were pleasantly surprised that the Pellet Reasoner performed sufficiently well even with larger ontologies. However, at the current time, the overall approach for modeling in-car infotainment systems based on ontologies presented in [1] requires significant amounts of time when querying the ontology for inferred knowledge. Hence, we advise to carefully assess the necessity of each class in an ontology due to the potential performance impact.

Guideline 2: Divide and conquer.
If the context-aware in-car infotainment system depends on a large amount of contextual information that is the basis for reasoning, an ontology *and*, there exist distinct groups of contextual information without any connections between them, it is well advised to avoid modeling a single monolithic ontology. To improve the performance of reasoning, the ontology should be split up over multiple constituent ontologies that contain elements with particularly high cohesion but only reference the ontologies that contain elements with less cohesion (in respect to the first ontology). Especially on a multi-core hardware, the overall system

performance would benefit, as the Pellet Reasoner currently does not to seem to automatically optimize reasoning for systems with more than one CPU core. For context-aware in-car systems, a suitable structuring of ontologies might be aligned with the aforementioned distinction of contextual information regarding the driver, the car and the environment.

Guideline 3: Feel free to use SWRL rules.
Our benchmark results show that SWRL rules only have a marginal impact on the performance of an ontology-based context-aware in-car infotainment system. In consequence, this means that the approach presented in [1] can be used to build high-performance and expressive ontologies with rules. Especially in our automotive domain it is beneficial to have a range of expressive possibilities without a loss of performance.

Following these guidelines, the modeler of a context-aware in-car infotainment system may capitalize on ontologies with rules for a both powerful and high-performance system.

6 Related Work

Several different types of context models can be found in the literature. An overview is provided in the survey paper by Strang and Linnhoff-Popien [5] as well as the survey paper by Bettini et al. [2]. Furthermore, benchmarks of ontologies can be found in literature. A well-known benchmark for ontologies including ontology generation is *LUBM* [8]. However, from our point of view, *LUBM* lacks support for generated random ontologies. Furthermore, *LUBM* does not support generating SWRL rules, which are crucial in our industrial practices. Weithöner et al. derived requirements for future benchmarks to make them more useful for developers of ontology-based systems [9]. There is also work in the literature that claims that current reasoners are not sufficiently fast for the intended use cases [10] and work that tries to optimize current reasoners [11]. More recently, there has also been work on how to predict the performance of a certain ontology [12]. We think, that predicting the performance can give a good indication but will not be a substitute for a benchmark as presented in this paper.

7 Conclusion

In this paper, we introduced a benchmark infrastructure to inspect the performance of modeling context-aware systems using ontologies. We presented a generator for plausible ontologies and associated source code, which both respect design principles derived from industrial practice. We performed multiple benchmarks on various different ontologies and derived guidelines for the design of ontology-based context-aware systems. With these guidelines, we intend to foster the adoption of ontologies by providing means for modeling and implementing high performance context-aware systems based on ontologies, such as in-car infotainment systems.

In our future work, we will optimize querying ontologies for inferred knowledge, as this seems to be the major issue when aiming for a high-performance ontology-based context-aware system. In addition, we will investigate other properties of ontologies that potentially have an impact on the overall system performance, such as the depth of the inheritance hierarchy in the ontology. Finally, we will evaluate the accuracy of the benchmark results within an industrial setting.

References

1. D. Lüddecke, N. Bergmann, and I. Schaefer, "Ontology-based modeling of context-aware systems," in *Proceedings of the 17th International Conference on Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, vol. 8767, pp. 484–500.
2. C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.
3. H. Winner, S. Hakuli, and G. Wolf, *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*, 1st ed. Vieweg+Teubner, 2009.
4. J. Schäuffele and T. Zurawka, *Automotive Software Engineering*, 5th ed. Springer Vieweg, 2013.
5. T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Proceedings of the 6th International Conference on Ubiquitous Computing. Workshop on Advanced Context Modelling, Reasoning and Management*, ser. Lecture Notes in Computer Science (LNCS), vol. 3205. Springer, 2004.
6. B. Motik, P. F. Patel-Schneider, and B. Parsia, "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition)," 2012. [Online]. Available: http://www.w3.org/TR/owl-syntax
7. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," 2004. [Online]. Available: http://www.w3.org/Submission/SWRL
8. Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 2-3, pp. 158–182, 2005.
9. T. Weithöner, T. Liebig, M. Luther, S. Böhm, F. v. Henke, and O. Noppens, "Real-world reasoning with OWL," in *The Semantic Web: Research and Applications*, ser. Lecture Notes in Computer Science, E. Franconi, M. Kifer, and W. May, Eds. Springer Berlin Heidelberg, 2007, vol. 4519, pp. 296–310.
10. C. Lee, S. Park, D. Lee, J.-w. Lee, O.-R. Jeong, and S.-g. Lee, "A comparison of ontology reasoning systems using query sequences," in *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, W. Kim and H. J. Choi, Eds., 2008, p. 543.
11. T. Wang and B. Parsia, "Ontology performance profiling and model examination: First steps," in *Proceedings of the 6th International Semantic Web Conference*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4825, pp. 595–608.
12. Y.-B. Kang, Y.-F. Li, and S. Krishnaswamy, "Predicting reasoning performance using ontology metrics," in *Proceedings of the 11th International Semantic Web Conference*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7649, pp. 198–214.