A Systematic Comparison of Semantic Integration Data Storage Architectures for Multidisciplinary Systems

Estefanía Serral, Olga Kovalenko, Thomas Moser and Stefan Biffl
Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems"
Vienna University of Technology, Vienna, Austria

Keywords: Multidisciplinary Projects, Data Integration, Ontologies, Querying Across Disciplines.

Abstract: Multidisciplinary projects typically rely on the contributions of various disciplines using heterogeneous engineering tools. This paper focuses on the challenge of querying across different disciplines, which may be influenced by the selection of a proper instance data storage architecture for storing the heterogeneous tool data. Specifically, we have identified three different architectures: ontology file stores, triple stores and relational database stores. This paper systematically compares these architectures using an industrial case study and analyses their selection according to important requirements such as performance and maintainability.

1 INTRODUCTION

Multidisciplinary projects bring together experts from various engineering domains and organizations that work in a heterogeneous engineering environment. This environment involves a wide range of models, processes, and tools that were originally not designed to cooperate seamlessly. In order to reach the common goal of developing software products in the engineering team, it is important to share the necessary knowledge for common work processes between engineering-domain experts. These experts usually want to use their well-known local tools and data models, and additionally want to access data from other tools in their local syntax. Thus, experts have to invest considerable effort to bridge the semantic gaps between common project-level engineering concepts and the diverse local data representation.

In this context, the three major challenges of semantic data integration in the area of multidisciplinary projects can be defined as (a) the definition of mappings between local and common engineering concepts for integrating and sharing the necessary data; (b) the transformations between local engineering concepts used in the different domains following these mappings; and (c) queries to local engineering concepts using the syntax of the common engineering concepts. The first two challenges have already been addressed in recent research (Moser et al., 2011)(Moser and Biffl, 2012) by proposing a semantic data integration framework, the so-called Engineering Knowledge Base (EKB). The EKB maps the data elements of local tool data models (models of tools which are relevant for supporting specific engineering tasks), to the respective elements in a common project-wide or domain-wide data model (Moser and Biffl, 2012), so called the Engineering Object (EO) Model. The EKB models the tool data models and the common data model using ontologies and explicitly represent the mappings using a machine-understandable ontology syntax.

This paper focuses on the third challenge, which is the ability of performing queries in a general project context and independently of local engineering tools. To query the knowledge of the local tool data models using the EKB, we apply the mediator architecture(Wiederhold, 1992). Mediated query systems represent a uniform data access solution by providing a single access point (so called common model) for querying various data sources. A mediator contains a global query processor which is used to send subqueries to local data sources. The local query results are then combined and returned back to the query processor. Using the EO ontology model (the common model), the structure of the query is more intuitive for the user because it corresponds more to the users appreciation of the project relevant information. We use SPARQL(Pérez et al., 2006) for describing a query over the EO ontology (i.e., the mediator) since it is the W3C standard for querying ontologies. SPARQL syntax makes virtually all join operations implicit, making the queries more compact and easier to describe

Serral E., Kovalenko O., Moser T. and Biffl S..
A Systematic Comparison of Semantic Integration Data Storage Architectures for Multidisciplinary Systems.
DOI: 10.5220/0004318101900197
In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2013), pages 190-197
ISBN: 978-989-8565-42-6

and to get them right with less debugging time spent. This SPARQL query is decomposed and rewritten in order to be executed over the local tool data models. The SPARQL queries are locally evaluated and the results are returned to the mediator site.

With a specific focus on querying in the EKB, different factors may have an impact on important project requirements (such as scalability, maintenance, semantic expressiveness, etc.). In this paper, we focus on the study of three different semantic integration data storage architectures that can be used in the EKB for storing the instance data (i.e., the individuals). The paper presents a comparative study of these architectures, which can be classified as follows:

- Ontology file stores: the ontology definition and the data instances are stored using file systems based on ontology languages.
- Triple stores: the ontology definition and the data instances are specified using an ontology language, but the data instances are internally stored using special ontology-based databases capable of storing triples (i.e., subject-predicate-object expressions, which is the specific form of describing data using an ontology).
- Relational databases stores: the ontology definition is defined in an ontology language, but the data instances are stored in relational databases.

The remainder of this paper is structured as follows: Section 2 pictures a typical multidisciplinary case study, which is situated in the hydro power plant engineering domain. Sections 3 - 5 present the three introduced architectures for data storage. Section 6 describes how the data of the case study can be queried depending on the applied data storage architecture. Further, these architectures are discussed and compared in Section 7. Finally, Section 8 concludes the paper and identifies further work.

2 CASE STUDY: HYDRO POWER PLANT ENGINEERING

A typical example of multidisciplinary system is the engineering of hydro power plants. Figure 1 shows two engineering tool data models corresponding to two different domains: software engineering (SE) domain and mechanical engineering (ME) domain. These tools contain local data sources, which produce and/or consume data with heterogeneous data structures. Specifically, the data model of the SE domain contains information about Programmable logic controller (PLC) variables; and the data model of the ME domain comprises information about monitoring devices. Figure 1 shows a simplified version of these models. The left hand side shows the concept *Sensor* from the ME domain, while the right hand side shows the concept *Variable* from the SE domain. The attribute *Type* of the concept *Sensor* could be defined as either *analog* or *digital*, which directly correlates to the attribute *Type* of the concept *Variable* that could be defined as either *float* or *boolean*. These two concepts can be mapped to the common engineering object *Signal* which is shown in the middle of the figure. The upper side of the figure shows some instances of these data models.

Listing 1: Mapping M1.

```
ME:Sensor(?sensor) ∧
ME:hasID(?sensor, ?sensor_id) ∧
SE:Variable(?var) ∧
SE:deviceID(?var, ?sensor_id) →
Signal(?signal) ∧
:corrVariable(?signal, ?var) ∧
:corrSensor(?signal, ?sensor)
```

The EKB facilitates the efficient data exchange between these tools by defining the tool data models and the EO model using ontologies and making explicit and in machine-understandable way the mapping among them. The EO in this system is identified as the *Signal* concept, which is composed by: a variable, a sensor, and a property to indicate whether the signal is consistent or not. Thus, the Signal EO links the two domain-specific data models (i.e., the ME ontology and the SE ontology).

For representing the mappings that link the local tool ontologies with the EO ontology, we specify that the properties *corrVariable* and *corrSensor* of the Signal EO are object properties; and that their range is *Variable* (from the SE ontology) and *Sensor* (from the ME ontology), respectively. In addition, we use SWRL rules[1]. In particular, three mappings are defined. The first mapping (see Listing 1) defines that if a variable is linked with a sensor then there must be a corresponding signal in the EO ontology.

The next two mappings (see Listing 2 and Listing 3) basically mean that if a value obtained from a monitoring device *sensor* is represented in PLC code as a variable *var* then the types of these two must conform to each other (”digital” and ”boolean”; ”analog” and ”float”). Otherwise there is an inconsistency in the signal description that has to be checked by domain experts.

Next we describe the different semantic integration data storage architectures that can be applied in the EKB according to the store used for the engineer-

[1] http://www.w3.org/Submission/SWRL

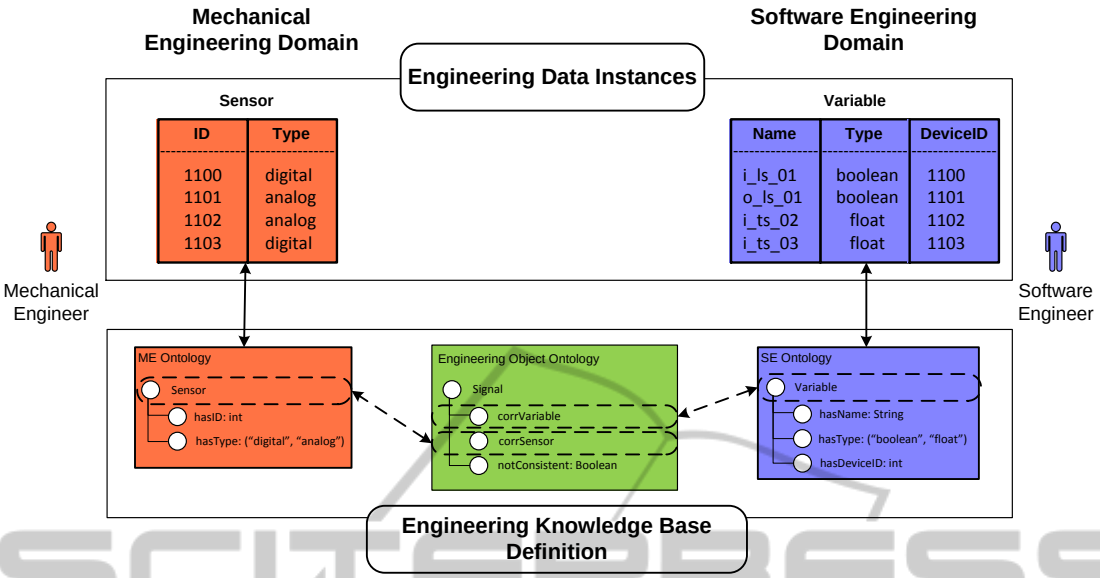

Figure 1: EKB for Automation Systems Engineering (adapted from (Moser and Biffl, 2012)).

Listing 2: Mapping M2.

```
SE:hasDeviceID(?var, ?sensor_id) ∧
SE:hasType(?var, "boolean") ∧
ME:hasID(?sensor, ?sensor_id) ∧
ME:hasType(?sensor, "analog") →
:notConsistent(?signal, true)
```

Listing 3: Mapping M3.

```
SE:hasDeviceID(?var, ?sensor_id) ∧
SE:hasType(?var, "float") ∧
ME:hasID(?sensor, ?sensor_id) ∧
ME:hasType(?sensor, "digital") →
:notConsistent(?signal, true)
```

ing data instances. Afterwards, we use the explained case study for showing how the data is queried in each one of the architectures.

3 USING ONTOLOGY FILE STORES

The engineering data instances can be directly stored as individuals together with the corresponding local tool ontology on XML-based semantic files, either in a single file or distributed among several segments across several files.

For managing and querying the data, this approach loads the whole ontology model data (i.e., the engineering knowledge base definition and the data instances) into the memory. This allows the data to be queried using SPARQL queries through the EO ontology, but also introduces high memory cost. For instance, the Jena framework[2], Oracle 11g[3] and Sesame[4] provide this type of store, charging the models in memory. The family of semantic RDF[5] (Resource Description Framework) storage solutions OWLIM(Bishop et al., 2011) also provides with SwiftOWLIM, which is an in-memory RDF database. It uses optimized indexes and data structures to be able to process tens of millions of RDF statements on standard desktop hardware. Jena, Sesame and SwiftOWLIM source code are provided free of charge for any purpose.

Some approaches, such as (Novák and Sindelár, 2011) (Battista et al., 2007) have successfully used this architecture; however, in both approaches the users plan to use other more sophisticated storage solutions to make their approaches scalable to large models. More detail about these approaches can be found in (Serral et al., 2012).

This type of store is considered very useful for tests or small examples, but in general it is not recommended for working with large models.

4 USING TRIPLE STORES

In the same way that the engineering data instances can be stored in files and managed in memory, they can be also stored and managed using triple stores

[2]http://jena.apache.org
[3]http://www.oracle.com
[4]http://www.openrdf.org
[5]http://www.w3.org/RDF/

(one triple store for each tool ontology). In a triple store, the local tool ontologies and the instances are specified using ontology languages; however, the instances (i.e., the individuals) are internally managed using special databases built specifically for storing triples. These databases are also called semantic stores or semantic web databases. In this way, the database management is transparent for users and the data can be queried using SPARQL queries through the EO ontology.

The generic schema of these special databases corresponds to one table that contains three columns named Subject, Predicate and Object. Thus, it reflects the triple nature of RDF statements. The triple store can be used in its pure form (Oldakowski et al., 2005), but most existing systems add several modifications to improve performance or maintainability. A common approach, the so-called normalized triple store, is adding two further tables to store resource URIs and literals separately, which requires significantly less storage space (Harris S, 2003). Furthermore, a hybrid of the the normalized triple store can be used, allowing the values to be stored themselves either in the triple table or in the resources table[6].

By using this approach, users can manage the data in an ontology language (e.g., OWL or RDF) and use SPARQL queries having a better performance than ontology file stores thanks to the use of the databases.

Some relevant examples of these stores are TDB[4] and BigOWLIM (Lu et al., 2007). The TDB component is provided by the Jena framework for optimized RDF storage and query. TDB supports the full range of Jena APIs and TDB performs and scales well. BigOWLIM is designed for large data volumes and uses file-based indices that allow it to scale, positioning it as an enterprise-grade database management system that can handle tens of billions of statements.

Some examples of applications of this architecture are (Klieber et al., 2009)(Miles et al., 2010). Both of them have used Jena TDB triple store and specifically, (Klieber et al., 2009) shows the feasibility of using TDB with a population of about 4.5 million triples. More detail about these approaches can be found in (Serral et al., 2012).

5 USING RELATIONAL DATABASE STORES

Applying this architecture, the local engineering tool ontologies are specified using an ontology language, while the engineering data instances are stored using relational databases (one relational database for each tool ontology). In this case, only ontology classes, their hierarchies, object and data properties, axioms and restrictions are extracted into a memory. Instances have to be accessed by queries to the databases. Thus, the SPARQL queries written for querying the EO ontology have to be finally translated into the query language associated with the database.

This process can be defined as follows: to query the overlapping engineering concepts described in the EO ontology, a SPARQL query is specified. This query is then automatically transformed to the terms of the engineering tool ontologies, which include concepts that are mapped to the concepts of the common ontology that were included in the original query. These engineering tool ontology-specific queries are then executed using the query language of the database where the knowledge is stored, and the results are obtained. Then, these results are again transformed into their representation in the EO ontology by exploiting the mappings between tool ontologies and the EO ontology. Finally, the combined results are returned using the representation described in the EO ontology.

Several relational databases have been already proposed for applying this architecture. For instance, the Jena framework provides the SDB[4] component that allows the data of the model to be stored in a relational database. The storage is provided by a SQL database and many databases, such as Oracle, PostgreSQL, MySQL and MS SQL, are supported. A SDB store can be accessed and managed with the Jena API and can be queried using SPARQL. SDB is able to perform well up to 100 million triples.

Another example is D2RQ[7], which is an RDF based platform that is used to access the content of relational databases without having to replicate it into an RDF store. The D2RQ is open source software published under the Apache license.

Minerva(Zhou et al., 2006) is a component of the IBM Integrated Ontology Development Toolkit (IODT). The query language supported by Minerva is SPARQL. Using Minerva, one can store multiple large-scale ontologies in different ontology stores, launch SPARQL queries and obtain results listed in tables or visualized as RDF graphs. Currently, Minerva can take IBM DB2, Derby[8] and HSQLDB[9] as the back-end database.

Other examples are Oracle 10g RDBMS[5], Sesame

[6]Jena2 Database Interface - Database Layout. http://jena.sourceforge.net/DB/layout.html

[7]http://d2rq.org

[8]http://incubator.apache.org/derby

[9]http://www.hsqldb.org

on PostgreSQL[10], and DLDBOWL [11].

This type of storage adopts binary tables for the database, mapping the triples of the RDF graph to these binary tables. The most common schema is composed by a table for each class (resp. each property) in an ontology; each class table stores all instances belonging to the same class and each property table stores all triples which have the same property (Lu et al., 2007).

This architecture has been successfully applied in several projects such as (Calvanese et al., 2011)(Wiesner et al., 2011)(Tinelli et al., 2009). More detail about these approaches can be found in (Serral et al., 2012).

6 APPLYING THE DATA STORAGE ARCHITECTURES TO THE CASE STUDY

Using the EKB approach, comprehensive queries against the project data can be done in terms of EOs, i.e. using the classes and properties defined in the EO ontology. To be evaluated over the engineering data instances, which are located in data storages, such queries must be transformed. First, the initial query (in terms of EO ontology) must be rewritten in terms of the engineering tool ontologies. The rewriting process bases on mappings that bind the EO ontology with the engineering tool ontologies.

Listing 4: Query Q1.

```
SELECT ?signal
WHERE {
        ?signal notConsistent true;
}
```

In the hydro power plant engineering case study described in Section 2, two different engineering tool ontologies are integrated using the *Signal* EO. Let's consider that project engineers want to obtain a list of all signals that are not consistent. Such kinds of queries can be expressed in SPARQL as shown in Listing 4 (in terms of engineering object ontology).

Based on mappings M1, M2 and M3 (see Listings 1, 2 and 3) the query Q1 can be rewritten in terms of engineering tool ontologies, resulting in the two queries Q2 (see Listing 5) and Q3 (see Listing 6).

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

[11]http://swat.cse.lehigh.edu/downloads/dldb-owl.html

Listing 5: Query Q2.

```
SELECT ?var, ?sensor
WHERE {
     ?var  a SE:Variable;
     ?var  SE:hasDeviceID ?sensor_id;
     ?var SE:hasType "boolean";
     ?sensor a ME:Sensor;
     ?sensor ME:hasID ?sensor_id;
     ?sensor ME:hasType "analog";
}
```

Listing 6: Query Q3.

```
SELECT ?var, ?sensor
WHERE {
     ?var a SE:Variable;
     ?var SE:hasDeviceID ?sensor_id;
     ?var SE:hasType "float";
     ?sensor a ME:Sensor;
     ?sensor ME:hasID ?sensor_id;
     ?sensor ME:hasType "digital";
}
```

If the engineering tool data instances are stored in ontology file stores or triple stores, then Q2 and Q3 can be already executed to obtain results since SPARQL queries can be executed across several ontology models. However, if the data is stored in databases then several further transformations will be needed before the queries could be executed over the data. Basically, Q2 and Q3 must be translated into SQL queries to be evaluated over the databases. This can be done in 2 steps. First of all, since each query has terms from more than one database, evaluating them independently over SE or ME database would fail. This problem can be solved by creating a set of independent queries.

For the sake of brevity we show only the rewriting for Q2 (as it will be similar for Q3). Listing 7 shows the rewritten SPARQL query for the SE engineering tool ontology, while Listing 8 shows the rewritten SPARQL query for the ME engineering tool ontology.

Listing 7: Query Q4.

```
SELECT ?var, ?sensor_id
WHERE {
     ?var a SE:Variable;
     ?var SE:hasDeviceID ?sensor_id;
     ?var SE:hasType "boolean";
}
```

Listing 8: Query Q5.

```
SELECT ?sensor, ?sensor_id
WHERE {
     ?sensor a ME:Sensor;
     ?sensor ME:hasID ?sensor_id;
     ?sensor ME:hasType "analog";
}
```

These independent queries can be finally translated to SQL and evaluated over the domain databases as shown in Listing 9 and Listing 10.

Listing 9: Query Q6.

```
SELECT *
FROM Sensor s
WHERE s.hasType = "analog"
```

Listing 10: Query Q7.

```
SELECT *
FROM Variable v
WHERE v.hasType = "boolean"
```

After obtaining the results, an intermediate join should be done to obtain the correct answer for the initial query.

7 DISCUSSION

The comparison among the presented methodologies is summarized in Table 1. Based on the consulted literature, the following aspects have been analyzed and compared:

- ***Query and Result Transformations.*** It indicates if a SPARQL query can be directly executed or it has to be transformed to other query languages to be executed. If the query has to be transformed, then another transformation is also needed to return the results as asked in the SPARQL query.

  Ontology files and triple stores allow SPARQL queries to be directly executed across the ontology models. However, the use of relational databases requires: 1) the SPARQL queries to be transformed to the corresponding relational query language; and 2) the obtained results from the databases to be transformed in accordance to the data asked in the SPARQL query.

- ***Scalability.*** It indicates how efficient the architecture for accessing the data is (the more response time, the less efficiency) and how it scales to large data applications.

  Ontology files are very efficient for small models greatly reducing the load and update time; however, when the data grows in volume, this storage becomes unsuitable (Shen and Huang, 2010)(Vysniauskas et al., 2011).

  The performance of the relational database methodology considerably varies according to the used database (Shen and Huang, 2010); however, this methodology provides many query optimization features, thereby contributing positively to query response time (Lu et al., 2007). According to the Berlin SPARQL Benchmark (Bizer and Schultz, 2009), the comparison of the fastest triple store with the fastest relational database store shows that the last one has a better overall performance with increasing dataset size.

- ***Reusability of Existing Knowledge.*** It indicates the facilities provided for reusing data instances stored in other existing databases or ontologies.

  Nowadays, there is a massive amount of data stored in SQL databases with associated technology, infrastructure and know-how (Sami Kiminki and Hirvisalo, 2010). The use of relational databases facilitates to reuse this data. However, in a similar way, ontology files and triple stores facilitate the reuse of data stored in a compatible ontology language.

- ***Support for SQL Queries***: Queries can be performed over the ontology (high level of abstraction), but also directly over the database (lower level of abstraction).

  Only the use of relational databases supports SQL queries; in this way, users and applications can perform queries at both ontology level (higher level) and database level (lower level).

- ***Facilities to Use Semantic Technologies.*** The use of semantic standard languages like OWL or RDF facilitates the use of numerous semantic technologies available to perform tasks such as data management (e.g., by using Protégé or Jena), reasoning (using reasoners such as Pellet, Racer[12], etc.), ontology mapping or model transformation. For instance, given the mapping between a source ontology and a target ontology, the OntoMerge (D. Dou and Qi, 2003) tool can translate instances that conform to the source ontology to instances conforming to the target ontology.

  Ontology files and triple stores represent the data using semantic standard languages; therefore, these methodologies facilitate the use of existing semantic technologies.

- ***Maintenance.*** Facilities provided in order to perfect the system, to adapt the system and to correct the system (Lientz and Swanson, 1980).

  In ontology files and triple stores both the knowledge base definition and the data instances can be maintained using semantic tools (e.g., Protégé) and middlewares (e.g., Jena). Using relational databases, the knowledge base definition and the data instances are managed differently. While the ontologies' definition can be managed by using

[12]http://www.racer-systems.com/

Table 1: Comparative table of the presented data storage methodologies.

	Ontology Files	Triple Stores	Relational Databases
Query and Result transformations	Not needed	Not needed	Needed
Scalability	Low	Medium/High	High
Reusability of existing data	Facilities to reuse data stored in a compatible ontology language	Facilities to reuse data stored in a compatible ontology language	Facilities to reuse data stored in relational databases.
Support for SQL queries	Yes	No	Yes
Facilities to use semantic technologies	Yes	Yes	Only for the knowledge base definition
Maintenance	Using semantic tools.	Using semantic tools.	Knowledge base definition: using semantic tools. Database schema has to be synchronized with the ontologies when they change. Data instances: using relational database tools.
Semantic expressiveness	High	High	Low
Available tools	SwiftOWLIM, Jena framework, Oracle 11g, Sesame, etc.	Jena TDB, BigOWLIM, etc.	Jena SDB, D2RQ, Minerva, Oracle 10g, RDBMS, Sesame on PostgreSQL, and DLDBOWL, etc.

semantic tools, the data instances have to be managed using relational database tools, at a lower level of abstraction. In addition, the schema of the databases has to be modified (e.g., deleting or creating tables) when ontologies change.

- ***Semantic Expressiveness*** It indicates if the architecture provides total support for representing semantics.

  Since ontology files and triple stores use semantic languages for representing the data instances, they can be semantically represented at a high level of abstraction (using the concepts defined in the ontologies, which are close to the domain); therefore, they provide more semantic expressiveness than relational databases, where semantics may be lost in the process for transforming the data into a relational schema (Uschold and Gruninger, 2004).

- ***Available Tools.*** It gives some examples of current available tools for applying each one of the architectures.

8 CONCLUSIONS AND FURTHER WORK

In this paper, we have performed a systematic comparison of three different data storage architectures with a specific focus on querying data over heterogeneous engineering tools. We have summarized available technologies that make these architectures possible and also research approaches that have applied these technologies successfully.

The comparison shows that the data storage selection is an important architectural decision that must be made according to the requirements of the software project to develop. Thus, ontology file stores are better for testing and for small data models; however, they are not scalable for large models, for which triple stores and relational databases are more appropriate.

The management of the data in triple stores is quite efficient and invisible for the users. Furthermore, these stores provide more semantic expressiveness and allow SPARQL to be directly executed over the data instances. In addition, triple stores and ontology file stores allow the use of semantic web tools facilitating the reusability of existing data represented in compatible ontology languages. On the other side, the use of relational database stores provides a very good performance for accessing, managing and querying the data of large models. In addition, this type of store facilitates reusability of existing knowledge in SQL databases.

Besides the data store, other factors have an influence in the aspects discussed in Section 7. For instance, storage layouts and the order of query patterns

have significant effects on query performance (Shen and Huang, 2010). As further work, we plan to evaluate these effects in our case study.

ACKNOWLEDGEMENTS

This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

REFERENCES

Battista, A. D. L., Villanueva-Rosales, N., Palenychka, M., and Dumontier, M. (2007). Smart: A web-based, ontology-driven, semantic web query answering application. In *Semantic Web Challenge*, volume 295. CEUR-WS.org.

Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2011). Owlim: A family of scalable semantic repositories. *Journal of Web Semantics*, 2(1):3342.

Bizer, C. and Schultz, A. (2009). The berlin SPARQL benchmark. *Int. J. Semantic Web Inf. Syst*, 5(2):1–24.

Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., and Savo, D. F. (2011). The mastro system for ontology-based data access. *Semantic Web*, 2(1):43–53.

D. Dou, D. M. and Qi, P. (2003). Ontology translation on the semantic web. In *Proceedings of International Conference on Ontologies, Databases and Applications of Semantics*.

Harris S, G. N. (2003). 3store: Efficient bulk rdf storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems, PSSS 2003*.

Klieber, W., Sabol, V., Muhr, M., and Granitzer, M. (2009). Using ontologies for software documentation. In *Proceedings of Malaysian Joint Conference on Artificial Intelligence*.

Lientz, B. P. and Swanson, E. B. (1980). *Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley.

Lu, J., Ma, L., 0007, L. Z., Brunner, J.-S., Wang, C., Pan, Y., and Yu, Y. (2007). Sor: A practical system for ontology storage, reasoning and search. In *VLDB*, pages 1402–1405. ACM.

Miles, A., Zhao, J., Klyne, G., White-Cooper, H., and Shotton, D. M. (2010). Openflydata: An exemplar data web integrating gene expression data on the fruit fly drosophila melanogaster. *Journal of Biomedical Informatics*, 43(5):752–761.

Moser, T. and Biffl, S. (2012). Semantic integration of software and systems engineering environments. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(1):38 –50.

Moser, T., Biffl, S., Sunindyo, W., and Winkler, D. (2011). Integrating production automation expert knowledge across engineering domains. *International Journal of Distributed Systems and Technologies (IJDST), Special Issue on Emerging Trends and Challenges in Large-Scale Networking and Distributed Systems*, 2(3):88–103.

Novák, P. and Sindelár, R. (2011). Applications of ontologies for assembling simulation models of industrial systems. In *OTM Workshops*, pages 148–157.

Oldakowski, R., Bizer, C., and Westphal, D. (2005). Rap: Rdf api for php. In *Proceedings of Workshop on Scripting for the Semantic Web, SFSW 2005, at 2nd European Semantic Web Conference, ESWC 2005*.

Pérez, J., Arenas, M., and Gutierrez, C. (2006). Semantics and complexity of sparql. In Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., and Aroyo, L., editors, *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer Berlin / Heidelberg.

Sami Kiminki, J. K. and Hirvisalo, V. (2010). Sparql to sql translation based on an intermediate query language. In *Proceedings of 6th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2010)*.

Serral, E., Kovalenko, O., Moser, T., and Biffl, S. (2012). Semantic integration data storage architectures: A systematic comparison for automation systems engineering. Technical report, Institute of Software Technology and Interactive Systems. http://cdl.ifs.tuwien.ac.at/files/TechReportNo_TR2012.2.5.pdf.

Shen, X. and Huang, V. (2010). A framework for performance study of semantic databases. In *Proceedings of the International Workshop on Evaluation of Semantic Technologies (IWEST 2010)*.

Tinelli, E., Cascone, A., Ruta, M., Noia, T. D., Sciascio, E. D., and Donini, F. M. (2009). I.m.p.a.k.t.: An innovative semantic-based skill management system exploiting standard sql. In Cordeiro, J. and Filipe, J., editors, *ICEIS (2)*, pages 224–229.

Uschold, M. and Gruninger, M. (2004). Ontologies and semantics for seamless connectivity. *SIGMOD Rec.*, 33(4):58–64.

Vysniauskas, E., Nemuraite, L., and Paradauskas, B. (2011). Hybrid method for storing and querying ontologies in databases. *Electronics and Electrical Engineering*, 115(9).

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer*, 25(3):38 –49.

Wiesner, A., Morbach, J., and Marquardt, W. (2011). Information integration in chemical process engineering based on semantic technologies. *Computers & Chemical Engineering*, 35(4):692–708.

Zhou, J., Ma, L., Liu, Q., Zhang, L., Yu, Y., and Pan, Y. (2006). Minerva: A scalable owl ontology storage and inference system. In *ASWC*, pages 429–443.