OXDBS – Extension of a native XML Database System with Validation by Consistency Checking of OWL-DL Ontologies

Christoph P. Neumann
cpn@cs.fau.de

Thomas Fischer
tom@cs.fau.de

Richard Lenz
rl@cs.fau.de

Institute of Computer Science 6 (Data Management)
Friedrich-Alexander University
Erlangen, Germany

ABSTRACT

Native XML database systems provide mature technology for persisting XML data and documents. Ontologies are often represented as XML-based documents like OWL-DL ontologies which allow for semantic consistency checking by formal description logic. Artificial intelligence provides reasoners as IT-support for consistency checking. Currently there exists no native XML database system which integrates logic reasoning for semantic consistency as addition to syntactic schema validation. The OXDBS project integrates a reasoner into a native XML database system, thus, allowing to assert consistency of ontological data at the most basic tier in an application environment.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*enhancement*; H.2.m [**Database Management**]: Miscellaneous; J.3 [**Computer Applications**]: Life and Medical Sciences—*medical information systems*; K.6.4 [**Management of Computing and Information Systems**]: System Management—*quality assurance*

General Terms

Design, Human Factors

Keywords

Semantic Web and Databases, Logic and Databases, Medical Systems, Data Quality and Semantics

1. MOTIVATION AND CHALLENGES

The core task of a database management system (DBMS) is to provide means that support the preservation of database consistency [1]. Absolute correctness of data cannot be guaranteed. One of many reasons for that is that our means for specifying integrity rules are not expressive enough to model real world conditions precisely. Another reason is that only some types of faulty data can actually be detected automatically. Nevertheless, database management systems traditionally provide powerful mechanisms that contribute to preserve data quality in a database and help to prevent faulty data even to enter the database. This requires a precise definition of what is considered correct from a database perspective. Typically, this is done by enforcing Atomicity, Consistency, Isolation and Durability (the "ACID" properties) for database transactions. Transaction management is not only to preserve operational semantics or to provide fault tolerance but also the essential assumption is that database transactions preserve semantic database consistency. If the database only contains results of completed transactions, then the database is considered consistent; thus, the transaction programmer is responsible for consistency preservation. Semantic integrity constraints help to prevent faulty transaction programs to commit, which (at least to some degree) takes the burden of semantic consistency checking from the application programmer.

Nowadays, a large portion of commonly used data is not as strictly structured as in traditional relational database systems. XML[1] has emerged as the syntactic standard for representing semi-structured data. Along with its widespread use both native XML databases as well as adaptations of relational database management systems have been developed. These approaches typically allow for syntactic consistency checking. Semantic checking of XML documents, however, is not part of today's XML database systems, though this clearly would greatly contribute to database consistency, which is the traditional mission of a DBMS.

Ontology technologies and XML technologies have gained a common ground: with OWL[2] as an XML-based approach to ontologies. Although OWL has become popular as Semantic Web technology, it has been adopted by the healthcare community [2, 3, 4]. For example, the OWL-based ontology *foundational model of anatomy* [5] has over 75,000 concepts, being under development since 1994. Automated tools are essential to check the consistency of large ontologies. OWL-DL reasoners provide automated detection of inconsistencies, which are very difficult to identify manually by humans.

Native XML database systems provide mature technology for persisting XML data and documents. The goal of the *ontological XML database system* (OXDBS) is to integrate a semantic reasoner into a native XML-DBMS, thus asserting consistency of ontological data at the most basic software

[1]eXtensible Markup Language
[2]Web Ontology Language

tier in an application environment. The OXDBS is based on available free or open-source software. The idea of the project is to bridge between database technology and artificial intelligence technology and to generalize syntactic validation (XML Schema) and semantic consistency checking (OWL-DL) in an OWL-aware XML-DBMS architecture.

2. BACKGROUND

As a means of data integration, domain ontologies are used as unique semantic reference. A principle in the context of description logics is the differentiation between *terminological box* (TBox) and *assertional box* (ABox). Roughly speaking, the TBox contains "terms", which is synonymous to "types", "class", or "concepts". The TBox also contains relations between concepts and concepts. The ABox contains "assertions", which is synonymous to "instances", "objects", or "individuals". The ABox also contains relations between individuals and concepts.

If OWL is used as ontology language, the TBox essentially represents the domain ontology. A TBox can be reused for multiple ABoxes, e.g. from different institutions that share a common conceptualization. To ensure semantic compatibility in a system environment, capturing any data requires an ontological commitment [6], which is gained by representing facts in form of an ABox being consistent to the shared TBox. Yet, concerning first-order logic, the distinction between TBox and ABox is not significant. Still, from an inference implementation point of view, the complexity of the TBox can greatly affect the performance of a given decision-procedure, independently of the ABox. Again, from a human point of view, the distinction is important because the responsible actors for TBox and ABox are distinct in most real-world scenarios.

There are different kinds of description logics, which can roughly be distinguished by the operators that are allowed. There is an informal naming convention, in which the expressivity is encoded in the label for a logic. For example, OWL-DL corresponds to $\mathcal{SHOIN}^{(\mathcal{D})}$ description logics. In OWL, as it is derived from RDF[3], knowledge is represented in form of *triples*, which are subject-predicate-object expressions. OWL data types may range over RDF literals or simple types defined in accordance with XML Schema data types.

3. OBJECTIVES AND METHODS

This paper initially provides a description of a common data production scenario based on available ontology tools. The limitations of this real-world scenario will be discussed. The objective of the OXDBS approach is to put consistency checking universally into the data tier of the application environment in order to foster data quality by preventing inconsistent data and to reduce uncontrolled data redundancy.

As a proof-of-concept, the OXDBS is implemented by freely available XML database systems and OWL-DL reasoners. The integration of semantic consistency checking should not annihilate the syntactic validation facilities. The objective is that the database mode in which OWL-DL consistency checking is performed can be switched on by configuration, and if switched off the database must behave as without our extension. After evaluation and selection of available software components, both components are analyzed: The primary focus in XML database analysis is the extraction of the validation mechanics inside the database architecture. The focus in reasoner analysis is to determine any supported interfaces to the description logics system, for the purpose of component integration.

[3]Resource Description Framework

The final step is the implementation of the OWL-validation module and its integration into the selected database. There are two basic reasons for complexity of this task: There exist several external interfaces and query languages for XML database systems. Depending on the interface, there exist different approaches to the validation mechanics in the chosen native XML database. In addition, the error handling and notification depends on the different database interfaces. In order to achieve comprehensive OWL validation support a deep understanding of the XML database architecture had to be gained.

4. USE-CASE SCENARIO

Domain experts and knowledge engineers to formalize domain knowledge in form of a TBox use an ontology editor. Taking clinical environments as example, end-users like assistant medical technicians or even secretaries provide case data at run-time, forming an ABox. Healthcare IT infrastructure traditionally is a workstation environment and in case of OWL-based data representation, the data is organized in files.

The most prominent ontology editor is *Protégé*, which stems from medical informatics research [7] and Protégé supports OWL. Yet, Protégé is just one of an arbitrary set of possible XML editors for OWL editing, and hence only one source of OWL document change. If applications instead of humans generate data, they would be required to integrate reasoners for consistency checking each on their own.

The data production in clinical scenarios suffers from the fact that capturing data is operationally separated from checking the consistency in regard to the ontology. Data is captured by desktop editors and stored via file systems; yet, to check consistency requires to apply a reasoner in form of a separate tool, with most reasoners being applicable by command-line. The consistency checking is a task that is assigned to end-users, it is not integral part of the data storage and on that account, it is often left out. In fig. 1 an outline is provided: some end-user (the actor on the left side) stores new data but forgets to check the consistency. Problems occur if his or her change left the file inconsistent and another end-user (the actor on the right side) opens the file, edits it, and responsibly checks it. Although the second end-user's changes are correct, he gets errors, which will not be resolvable by him or her.

In an ideal scenario, the consistency checking is part of the storage system, as it is outlined in fig. 2. The editor does not use the file system but, for example, *put* and *get* commands to store the OWL contents in a central XML-DBS. The database integrates the consistency checking and prevents users from storing inconsistent data. Such prevents friction between end-users and prevents anomalies with concurrent changes.

In fact, some resolution that users instinctively apply to deal with concurrency issues is file replication, which leads into uncontrolled data redundancy, and problems of file synchronization. Notably, it is not handy for users to store files in XML databases just for synchronization purpose if the database does not perform consistency checks: Then each user would have to temporarily store files in his or her file

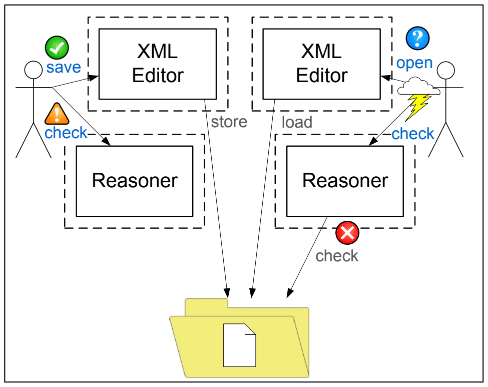

Figure 1: The real-world scenario

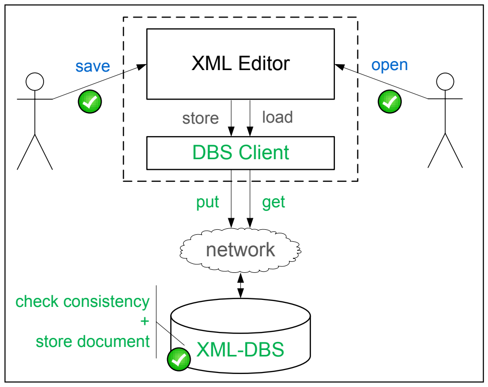

Figure 2: The idealized scenario

system just to apply the reasoner on command-line, before he or she commits the data to the database. The only feasible way, is to integrate consistency checking into the data tier of the application environment.

5. MATERIALS

Two basically required off-the-shelf components are 1) an XML database system and 2) a OWL-DL reasoner. The OXDBS focus lies on *native XML database systems* in contrast to XML enabled database systems.

The criteria catalog for the selection of a native XML database systems has been composed of: 1) open-source, 2) support for large-sized documents, 3) XQuery/XUpdate interface, 4) syntactic XML schema validation. The eXist [8] database system fulfills all criteria. The candidates BaseX and Apache Xindice were dismissed, because they do not include syntactic XML schema validation facilities. In addition, Xindice supports only small to medium sized documents. In conclusion, eXist in version 1.4 was selected as foundation for OXDBS.

The criteria catalog for an adequate reasoner is composed of: 1) free of charge, 2) provides consistency checking as black box functionality, 3) separated consistency checking of TBox and ABox is possible, 4) provides programming interface or network interface, and 5) supports all data types from XML Schema. The FaCT++ reasoner, implemented in C++, does not support XML Schema data types completely. Pellet reasoner, implemented in Java, and RacerPro, implemented in Lisp, both fulfill all criteria. They support all XML Schema data types and the DIG protocol [9], that is a standard interface for reasoning engines. In addition, RacerPro provides a powerful *New RacerPro Query Language* (nRQL). With `jracer`, there exists a Java API based on nRQL. In conclusion, all three candidate reasoners actually provide a mature implementation and adequate functionality; the RacerPro in version 1.9.0 was finally selected for OXDBS.

6. OXDBS

The OXDBS is a combination of a patched eXist DBS, the OWL validation module, and a RacerPro reasoner. An integral objective is that the semantic consistency checking can be switched on by configuration as a system extension, and that the former behavior of the database is preserved. Thus, the eXist configuration file must be complemented with an OWL validation mode. In addition, the information necessary to connect to the OWL-DL reasoner will also be configurable by the extended eXist configuration file.

For system extension, the validation mechanics inside the eXist architecture are analyzed. Unfortunately there are two separate validation facilities in eXist, which must both be adopted for OWL consistency checking. Both facilities will be described, as well as the OXDBS insertion of a delegation to a singular OWL validation module for both execution paths. A drawback emerged from the eXist internals, which prevents performant consistency checking only with the changed facts or concepts but requires the OXDBS extension to apply an expensive consistency checking with the complete ontological information, with all concepts and facts from TBox and ABox, for each data change.

Finally, the interaction between the OWL system extension inside the database and the OWL reasoner is described. Often TBox and ABox are stored in a single OWL document, but in an error case, being caused by an inconsistent document status, the reasoner provides information whether the inconsistency affects the TBox or the ABox. For the error handling, in regard to the transactional behavior, simple Boolean information is effective and the error origin in TBox vs. ABox is irrelevant. Nevertheless, for the end-user the TBox vs. ABox differentiation is very relevant in the case of an error. Unfortunately, the propagation of such error information is not possible with the available eXist architecture, because the validation return value is limited to a Boolean value.

6.1 Configuration of the Validation Subsystem

There are three different modes for the validation in eXist: *no*, *yes*, and *auto*. The *yes* mode is most restrictive: every file is tested for validity, implying that a file without schema information will always be rejected because validation is not possible. The *no* mode is least restrictive: every well-formed file is accepted. The *auto* mode tests files for validity if the file provides schema information but accepts files without schema information if they are well-formed. The mode cannot be changed during run-time.

The parsing of the eXist configuration file is based on an XML Schema file[4]. In order to extend the configuration, the attribute `mode` of element `validation` is just complemented with an enumeration entry `owl`. For generic configurabil-

[4]Being available as schema/conf.xsd in the eXist sources.

ity of the reasoner connection information, two attributes `ip` and `port` are added to the `validation` element. The class `org.exist.util.Configuration` parses the configuration file and translates the values into a hash map, which can be accessed by the subsystems at run-time.

In *owl* mode, the DBS will delegate every file for consistency checking to the new OWL validation module. Yet, the original behavior is preserved and the mode can be changed back to *yes/no/auto*. Besides the global configuration, the validation mode can be set for each collection individually[5]. Thus, it is possible to have all validation modes available at run-time and it seems good practice to use a dedicated collection for OWL databases.

6.2 XML Schema Validation in eXist

This section provides a short overview over the eXist validation facilities. There are several remote interfaces that are supported by eXist which allow to store XML content into the database, which are outlined in fig. 4. Support for XML-RPC and SOAP is implemented, both being flavors of XML-based remote invocation standards, in which the XML content is the payload as the parameter of a function call. The WebDAV and REST interface apply simple HTTP put and get of XML content, without an additional invocation wrapper, but instead use URL path segments that are dispatched in order to address XML databases. The eXist "AtomService" implements the *Atom Publishing Protocol* (AtomPub) and supports a binding of XML content in form of the *Atom Syndication Format* (ASF). Any of REST, XML-RPC, WebDAV, SOAP, and Atom interface accepts XQuery and XUpdate expressions: Both types of expressions can be put like content documents – with the difference that dedicated URL path segments are used to indicate XQuery and XUpdate service-endpoints that will interpret and execute the requests.

In addition it is possible to integrate the eXist database into applications as an embedded database. In this case, the standardized XML:DB API [10] for XML databases provides programmatic access.

Independent of the remote interface that is used to store XML content, the *implicit validation* of XML Schema is internally handled by the Collection.validateXMLResource(..). The various remote interfaces and their execution paths from the remote access layer to the Collection class are outlined in fig. 4. Whether or not eXist implicitly applies XML Schema validation is part of the configuration, as explained.

In contrast to the implicit validation, users are allowed to request *explicit validation* of stored XML content. Currently, only the XML:DB API and the XML-RPC of eXist provide this functionality; additionally, the XQuery standard also provides a validate command with an XML database locator as parameter. The execution paths for explicit validation are outlined in fig. 3. Independent of the technical origin of an explicit validation request, the internal class which has to handle such validation differs from implicit validation: The responsible class for explicit validation always is Validator with its validate(..) method, completely autonomous from the Collection class for implicit validation.

The common ground of Collection.validateXMLResource(..) and Validator.validate(..) is that both will use an XMLReader object from a SAXParser[6]. The basic idea in the OXDBS prototype implementation is now to substitute the SAXParser with a delegation to an OWL module that handles the interaction with an OWL-DL reasoner.

[5]The individual configuration for collections is applied by an collection.xconf file inside each collection.

[6]The eXist database relies on the Xerces XML parser for XML Schema validation.

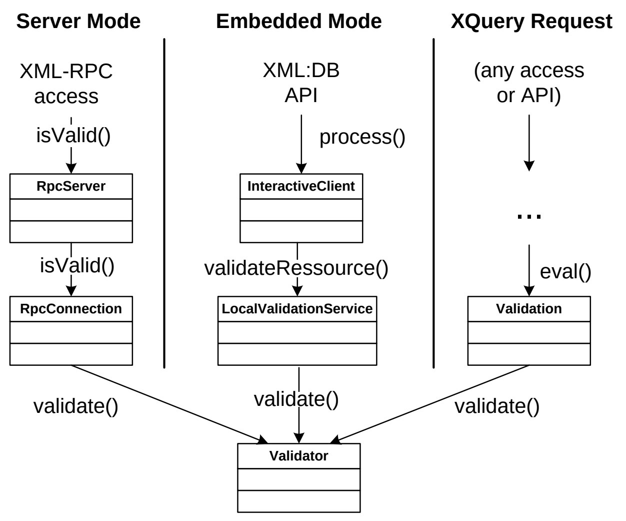

Figure 3: Execution paths for explicit validation

6.3 Limitations

There are several limitations in our OXDBS prototype implementation caused by eXist architecture limitations. The implicit validation only takes place if an XML document is stored into the database. If documents are changed by using XQuery-Update extension or XUpdate expressions the implicit validation is not performed.

The syntactic validation mechanism, being present in eXist, externalizes the whole XML document and delegates it to the XML Schema validator. The natural implementation of all OWL-DL reasoners provides sophisticated caching of ABox and TBox information to speed-up reasoning performance in the case of selective changes. On that account, reasoners provide interfaces to inform them only about changed concepts or individuals. However, because eXist only provides access to the complete XML document in the context of validation, the prototype externalizes the whole OWL document for delegation to the reasoner. The reasoner facilities can only be instrumented with full performance, if an OWL change set would be provided by an XML database to the OWL validation module.

6.4 OWL Consistency Checking Integration

The owlconsistency module handles the consistency checking of OWL-DL documents. The class OWLChecker is the facade of the module. Initially, the provided content in question is externalized by ExternalizedDocument into a temporary file as preparation for its delegation to a reasoner. Subsequently, the OWLChecker accesses the configuration information for `ip` and `port` in order to instantiate class OWLReasoner. In class OWLChecker, all technical exceptions which occur in the subsystem are handled and converted into EXistException of the eXist system. All exceptions that are caused by the indication of inconsistencies are converted into a Boolean conclusion as the return value of

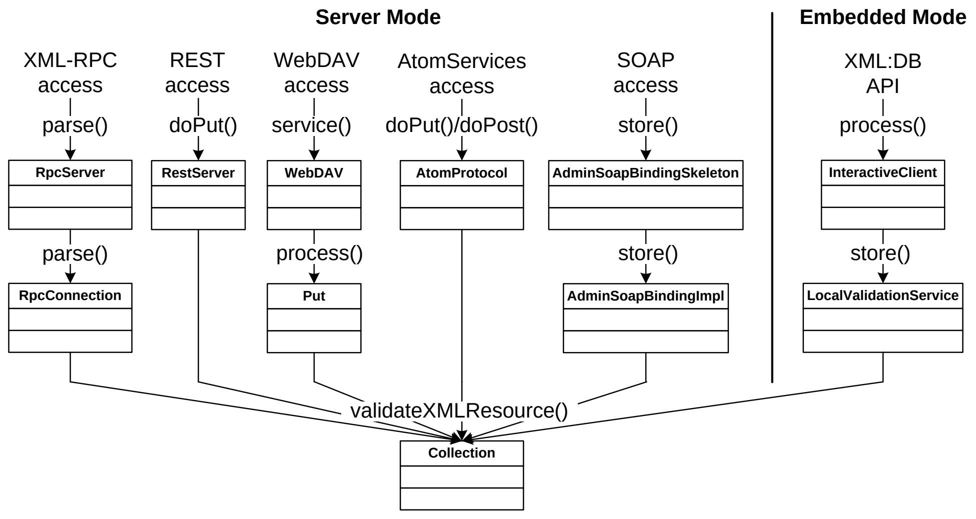

Figure 4: Execution paths for implicit validation

the OWLChecker.validate(..) method.

The class OWLReasoner is an abstraction of the actual reasoner. The current implementation uses the JRacer client library to access the services of a RacerPro server from Java. The JRacer class RacerServer provides the functionality to send nRQL commands to RacerPro. If a consistency error occurs, it is propagated as OWLException that allows to distinct ABox or TBox origin. The nRQL commands being successively sent are as follows:

1. `(full-reset)`
2. `(owl-read-file file)`
3. `(check-tbox-coherence)`
4. `(abox-consistent-p)`

The command `full-reset` invalidates any cached inference state of the reasoner. The `owl-read-file` does not implicate any consistency checks but RacerPro will initialize its internal data structures with the provided OWL contents. To begin the consistency checking, the `check-tbox-coherence` command will cause RacerPro to check the TBox. The return value contains a list of undecidable concepts; an empty list indicated by `NIL` is the happy case. In any other case, the check is aborted and an OWLException is thrown. If the TBox is consistent, the command `abox-consistent-p` will cause RacerPro to check the ABox against the TBox. The return value is either `T` or `NIL`; this time `NIL` indicates the inconsistent case. Again, an OWLException would be propagated. An OWL document is consistent, if and only if no OWLException is thrown. The OWLChecker facade translates the OWLException into a Boolean conclusion, as it is required by the eXist architecture.

7. RELATED WORK

A few similar approaches can be located in the literature and in the World-wide Web. However, an approach like the OXDBS with the focus on integration of an XML-DBS and OWL-DL consistency checking could hardly be found.

Triple stores [11] are databases for RDF triples. There are several implementations for triple stores, like Sesame, Jena TDB, or AllegroGraph. Yet, triple stores focus on providing querying facilities like SPARQL[7] [12] and do not provide OWL-DL inference and therefore no consistency checking. Besides, triple stores which have been built from scratch, like Jena TDB, are non-transactional and do not provide ACID properties.

Oracle Database, with its latest release of 11g, provides the *Semantic Technologies* framework [13] which supports RDF- and OWL-based data modeling, and query functionality like SPARQL support. Yet, Oracle does not integrate semantic consistency checking by a reasoner. Furthermore, Oracle is neither a native XML database system, nor does Oracle provide adoptable XML validation facilities.

The XTC [14] is a research prototype with a clean and modular architecture. Yet, it is not available as open-source. XQuery is available in XTC but not fully supported, neither is XUpdate. An integration of syntactic XML Schema validation is also not available. However, the XTC database is implemented very modular and hence the XTC promises a sound foundation for a reference architecture.

8. FUTURE WORK

If the assumption about database transactions, to preserve semantic database consistency, ought to be achieved for XML database systems, a reference architecture is required in which syntactic validation (XML Schema) and formal semantic reasoning (OWL-DL) is generalized in an autonomous validation subsystem. A flexible mechanism for *validation reporting* and error propagation is required, not only controlling the transactional behavior but also allowing the notification of neighbor systems about the validation or consistency problem. To integrate various validators or reasoners as a library, a *plug-in design* for the validation is necessary. An outline of such architecture is provided in fig. 5; the focus lies in the validation subsystem.

In order to improve validation performance, a reduced set of changes should be made available to the validation subsystem instead of the whole XML document. The *change set isolation* would depend on the validation technology and

[7]Recursive acronym for "SPARQL Protocol and RDF Query Language"

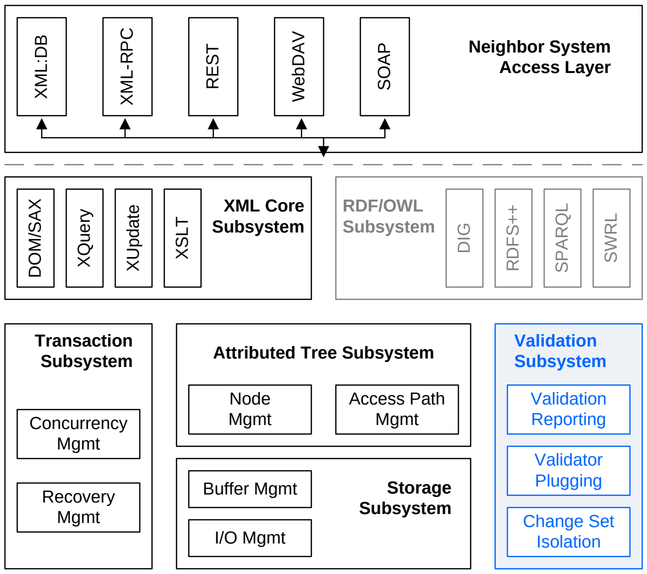

Figure 5: Reference architecture for a native XML-DBS with a generalized validation subsystem

would be different for XML Schema or OWL-DL. For OWL-DL purposes, any XML change needs to be set into the context of a changed XML structure for either a concept from the TBox or an individual from the ABox.

As a basis for such architecture, the eXist database is not adequate. The eXist implementation fails in regard to a modular architecture. Rather it is characterized by lots of code redundancy in which similar concepts are repeated in varieties of code and classes.

9. CONCLUSION

This paper has presented the OXDBS, which is a native XML-DBMS that is extended by a reasoner for OWL-DL data. A use-case scenario for healthcare infrastructure is provided which traditionally is a workstation environment. It requires the integration of a reasoner, beyond mere syntactic validation, to provide consistency checking as formal semantic validation of OWL-DL files being stored inside a database.

The OXDBS prototype is based on available free or open-source software and the OXDBS prototype is considered as a proof-of-concept. A criteria catalog is provided for native XML database systems and OWL-DL reasoners. Available software components have been evaluated. The eXist DBS as well as the RacerPro reasoner have been selected as a basis of the OXDBS prototype.

The OXDBS bridges a gap between XML database research and the field of artificial intelligence. The approach has several advantages. Both research domains have their own field of expertise, which are the ACID support for databases and high-performance inference for artificial intelligence. The OXDBS approach fosters their autonomous evolution but aims for integration standards. In the future, a reference architecture for native XML database systems is required, which defines the validation facility as an autonomous subsystem and formalizes the necessary interaction and dependencies to other XML-DBS subsystems as well as validation reporting in form of an open standard.

10. REFERENCES

[1] B. Bhargava and L. Lilien. Enforcement of data consistency in database systems. *Sadhana*, 11(1):49–80, 1987.

[2] Christine Golbreich and Ian Horrocks. The OBO to OWL Mapping, GO to OWL 1.1! In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[3] V. Kashyap and A. Borgida. Representing the UMLS semantic network using OWL. In *Proceedings of the 2nd International Semantic Web Conference*, pages 1–16. Springer, 2003.

[4] L.F. Soualmia, C. Golbreich, and S.J. Darmoni. Representing the MeSH in OWL: Towards a semiautomatic migration. In *Proceedings of the KR 2004 Workshop on Formal Biomedical Knowledge Representation*, pages 81–87, 2004.

[5] C. Golbreich, S. Zhang, and O. Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(3):181–195, 2006.

[6] Nicola Guarino. Formal ontology and information systems. In Nicola Guarino, editor, *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems (FOIS'98)*, pages 3–15, Trento, Italy, June 1998. IOS Press.

[7] J.H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson, N.F. Noy, and S.W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[8] Wolfgang Meier. eXist: An open source native XML database. *Web, Web-Services, and Database Systems*, pages 169–183, January 2003.

[9] Sean Bechhofer. The DIG Description Logic Interface: DIG/1.1. `http://dig.sourceforge.net/`, February 2003.

[10] Andreas Laux and Lars Martin. XUpdate working draft. `http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html`, September 2000.

[11] J. Broekstra, A. Kampman, and F. Van Harmelen. Sesame: An architecture for storing and querying RDF data and schema information. *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, page 197, 2003.

[12] J. Perez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *The Semantic Web – ISWC 2006*, pages 30–43, 2006.

[13] Oracle. Oracle Semantic Technologies Overview. `http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28397/sdo_rdf_concepts.htm`, September 2009.

[14] Theo Härder, Christian Mathis, Sebastian Bächle, Karsten Schmidt, and Andreas M. Weiner. Essential Performance Drivers in Native XML DBMSs. In *Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)*, volume 5901 of *LNCS*, pages 29–46. Springer, 1 2010.