

Towards ATIC: A Specialized Quadstore Tailored for Corporate Memories

Markus Schröder^{1,*}, Desiree Heim^{1,2}, Christian Jilek¹ and Andreas Dengel^{1,2}

¹Smart Data & Knowledge Services Dept., German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany

²Computer Science Dept., RPTU Kaiserslautern-Landau, Germany

Abstract

Corporate Memories are computer systems in organizations that continuously collect, update and structure knowledge for various tasks. The Resource Description Framework (RDF) is commonly used to represent such knowledge, which is stored in quadstores as subject-predicate-object-graph quadruples. Since typical RDF stores lack crucial features for enterprise scenarios, we propose an initial version of a tailored quadstore as part of a larger system called ATIC. Our implementation in this demo paper comprises RDF management with an SQLite backend, Discretionary Access Control on graphs and resources, confidence-attributed triples, change tracking with RDF patches and a virtual graph for accessing a file hosting service via RDF. This is accompanied with a server component providing endpoints and corresponding frontends. Results of a preliminary evaluation verify the correct operation of our prototype and indicate moderate performance.

Keywords

Resource Description Framework (RDF), Triplestore, Quadstore, Access Control, Corporate Memory

1. Introduction

The Resource Description Framework (RDF) [1] is a W3C standard for representing and linking resources primarily on the web using subject-predicate-object triples which are managed in triplestores. RDF is implemented in several programming language frameworks. One is the free and open source Java framework Apache Jena¹ which provides implementations of RDF and related Semantic Web standards. This includes TDB², a high performance store on a single machine used for storing and querying RDF data using the query language SPARQL [2]. It also supports RDF datasets (quadstores) and RDF-star (quoted triples) [3].

Using such RDF technologies, our research is typically centered around *Organizational* or *Corporate Memories*, which are computer systems in organizations that continuously collect, update and organize knowledge, providing it for various tasks like context-sensitive information delivery [4]. However, typical RDF stores (such as TDB) are designed with an open Semantic Web setting in mind. Consequently, they tend to lack features such as Identity & Access Management (IAM), Discretionary Access Control (DAC), provenance tracking and support for non-RDF data, making them not well suited to enterprise scenarios.

We therefore propose a first version of a quadstore tailored for use cases in Corporate Memories as part of a larger system called “ATIC” [5, p. 71] (more details in Section 3). This Java-based prototype implements the well-known Jena interfaces to ensure full compliance with RDF standards and compatibility with related software. It provides a built-in Discretionary Access Control (DAC) mechanism for sharing RDF graphs and resources among users and groups. As a storage backend, we utilize SQLite to adapt its database schema to our specific requirements. This demo paper presents an early prototype

Posters, Demos, Blue Sky, and Tutorials at SEMANTiCS 2026, Sep 2026, Ghent, Belgium

*Corresponding author.

✉ markus.schroeder@dfki.de (M. Schröder); desiree.heim@dfki.de (D. Heim); christian.jilek@dfki.de (C. Jilek); andreas.dengel@dfki.de (A. Dengel)

🌐 <https://www.dfki.uni-kl.de/~mschroeder/> (M. Schröder); <https://www.dfki.uni-kl.de/~heim/> (D. Heim); <https://www.dfki.uni-kl.de/~jilek/> (C. Jilek); <https://www.dfki.uni-kl.de/~dengel/> (A. Dengel)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹<https://jena.apache.org/>

²<https://jena.apache.org/documentation/tdb/>

which provides initial implementations for RDF storage and retrieval, access control, virtual graphs, RDF-star, and other functionalities. Its source code is publicly available on GitHub³ under the Apache License 2.0.

2. Related Work

In literature, several approaches have been proposed for RDF stores and SPARQL engines [6]. Their focus was on the efficient storage and retrieval of RDF data, often supported by specialized SPARQL query engines. In contrast, our system is designed to be extendable and feature-rich for Corporate Memory scenarios, with a secondary emphasis on performance. Therefore, our implementation uses the SPARQL processor provided by Jena.

Previous approaches have explored storing RDF data in Relational Database Management Systems (RDBMSs) as well. One notable store is Virtuoso [7], which also stores triples in relational tables. The system fully maps SPARQL queries to SQL queries and considers several optimization strategies. In Jena, SDB⁴ implements storage and retrieval of RDF triples on top of SQL databases. However, this sub-project has been retired and is in maintenance-only mode since 2013 (Jena version 3.17.0).

The combination of access control and RDF has also been discussed in literature [8]. Especially approaches implementing Discretionary Access Control (DAC) on top of RDF are related. Kirrane et al. [9] propose a system allowing to grant access for a SPARQL operation together with RDF graph patterns. In comparison, our DAC model allows to grant read or write access on RDF resource level. Jena provides a permission wrapper⁵ that intercepts calls to underlying RDF models. Since this access control system is not tightly integrated into an RDF store, it may introduce performance overhead. Other RDF stores like Virtuoso⁶ or GraphDB⁷ usually implement Role Based Access Control (RBAC) on named graph level.

As no RDF store currently available fulfills our requirements, we started the implementation of ATIC and present a first demonstration of an early system.

3. Demo System

ATIC is planned to be a feature-rich system for Corporate Memories [5, p. 71]. Its acronym is derived from its general features such as Access Control, Time-dependency, Individual-centric, and Context-focused. The name is also inspired by the English word *attic*, an architectural superstructure of a building, symbolizing the meta-model in Corporate Memories. Yet, in this paper, we present a preliminary version of a quadstore that implements only a subset of its intended features. Our demo page⁸ contains a link to a deployed instance for testing and also presents a video showing the main aspects in action.

The following section discusses the initial implementations of the developed features in detail:

RDF Management in SQLite. As a storage backend, we utilize SQLite⁹, a lightweight, serverless, easy to embed relational database. Our database schema consists of tables for Graph & Resource/Property storage (graph, resource, resource_uri, property), Quadruple storage (spog, sp1g), Access Control (graph_acl, resource_acl), RDF-star (resource_spo, resource_sp1) and User & Group Management. To ensure full compliance with RDF standards and compatibility with related software, the code implements Jena's `DatasetGraph` and `Graph` interfaces (Version 5.6.0). For the typical operations like add, remove, find, contains and size corresponding SQL queries are dynamically formulated and executed. For performance reasons, a long-lived connection is kept to the database to reuse prepared and cached SQL statements.

³<https://github.com/DFKI/ATIC>

⁴https://jena.apache.org/documentation/archive/sdb/sdb_index.html

⁵<https://jena.apache.org/documentation/permissions/>

⁶<https://docs.openlinksw.com/virtuoso/rdfgraphsecurity/>

⁷<https://graphdb.ontotext.com/documentation/11.3/access-control.html>

⁸<https://www.dfki.uni-kl.de/~mschroeder/demo/atic>

⁹<https://sqlite.org/>

Discretionary Access Control. Our approach is designed to be in line with typical sharing mechanisms in cloud applications: owners of folders or documents share these resources with other users or groups (principals) on certain permission levels. Analogously in our system, folders correspond to RDF graphs and documents correspond to RDF resources. We distinguish between four permission levels (least to the most permissive): READ (can view), REFER (can refer to), EDIT (can edit), ADMIN (can administer). Access to RDF graphs depends on the assigned permission level: READ permits listing of triples, EDIT permits triple modification, and ADMIN additionally permits graph deletion. Permissions assigned to RDF resources determine the permissions of the RDF triples associated with them: considering only the subject (s) and object (o) positions, the effective permission of a triple is given by $\min(\text{perm}(s), \text{perm}(o))$, ensuring that access to the triple is restricted by the less permissive of the two resources. Consequently, a triple can only be added or removed if both its subject and object have at least EDIT permission. For triples with a literal object, only the subject’s permission is considered. The special REFER permission is designed for ontologies: while users are not allowed to modify ontology concepts, they can refer to them (i.e. add/remove a triple) using a resource with at least EDIT permission. A creator of a new graph or resource is automatically assigned ADMIN permission to express ownership. A resource is considered to be created when it first appears as the subject or object of a newly inserted RDF triple.

Confidence-Attributed Triples. Using a relational database as the storage backend allows the system to be easily extended to meet additional requirements. One aspect concerns the representation of uncertainty in RDF triple statements. This is particularly relevant when AI agents autonomously modify RDF graphs, for instance, by adding extracted information [10]. Since such automatically generated statements may rely on incomplete evidence or uncertain inferences, it is desirable to explicitly capture the confidence associated with each statement. Building on the existing Jena interfaces, we introduce the class `AticTriple`, which is able to associate each triple with a floating-point confidence value in the range $[0, 1]$.

RDF Patch Emitter. To keep track of changes in our store, we utilize Jena’s RDF Patch implementation¹⁰. For each committed transaction in the system, a patch is generated capturing modified quadruples, user/group creation & assignments, and permission changes. For query performance reasons, the patches are stored in a separate SQLite database, enabling external services (e.g. full-text search) to synchronize with the current state of the data store.

Virtual Graphs. Corporate Memories often need to integrate legacy, non-RDF data sources. To avoid full materialization of these systems into RDF, we rely on virtualization through virtual graphs [11]. As an example in our demo, the file hosting service Nextcloud¹¹ is accessed via its WebDAV interface [12] using the Java WebDAV client Sardine¹². This graph implements Jena’s `Graph` interface to serve triples virtually per request.

An additional **Server** component has been developed using the web framework Javalin¹³. It provides an authentication layer with JSON Web Tokens (JWT), serves Vue.js¹⁴ based frontends and exposes various REST endpoints.

Registration & Login Endpoints. The system provides user and group management. A default admin user is created during the system’s initial startup, and additional users may subsequently be registered. Upon successful login, the authentication endpoint issues a JWT that is stored in an HTTP cookie.

Upload Endpoint. This interface supports the upload of RDF data in various serialization formats. In this process, permissions can be assigned to principals for resources contained in the imported data.

SPARQL Endpoint. For RDF retrieval a SPARQL [2] endpoint and corresponding user interface is provided. Internally, our system reuses Jena’s SPARQL processor¹⁵ (`jena-arq`).

Configurable CRUD Endpoints. To support diverse application requirements while minimizing

¹⁰<https://jena.apache.org/documentation/rdf-patch/>

¹¹<https://nextcloud.com/>

¹²<https://github.com/lookfirst/sardine>

¹³<https://javalin.io/>

¹⁴<https://vuejs.org/>

¹⁵<https://jena.apache.org/documentation/query/>

implementation effort, our application provides configurable Create, Read, Update, and Delete (CRUD) endpoints defined through YAML¹⁶ files. These files specify how operations are mapped to SPARQL queries and how responses are serialized as JSON-LD [13].

Molecule Endpoint. In web based user interfaces as well as Corporate Memory scenarios in general, it is usually more convenient to work with objects (documents) to visualize and interact with, instead of an unordered set of RDF triples. For a certain resource, we call its RDF graph excerpt a *molecule*, since it consists of incoming and outgoing triples as well as literals. Our endpoint provides several options for selection and projection to tweak a molecule’s JSON-LD [13] representation.

4. Evaluation

In a preliminary evaluation, our quadstore was assessed with respect to two aspects: correctness and performance. Both were evaluated independently.

To ensure the correct behavior of RDF storage and retrieval, we developed 18 JUnit test classes comprising a total of 145 test cases. Correct behavior is defined as passing all assertions in the JUnit test suite; thus, each test case is evaluated using a binary pass/fail criterion with no tolerated deviations from the expected result. Our test suite evaluates a range of functionalities, including basic SQLite database operations, RDF dataset usage, graph and resource sharing, SPARQL query evaluation, blank node handling, virtual graph loading, and RDF-star support. Additionally, we selected a subset¹⁷ of 18 Jena TDB2 JUnit test classes¹⁸ (Jena Version 5.6.0) to evaluate our prototype. The only modification made to the code was replacing TDB2 with our quadstore instance for test execution. In total, 240 tests passed and 29 were skipped, primarily because they depended on TDB2-specific functionality, such as internal implementation details, a special union default graph setting, automatic empty-graph deletion behavior, and transaction promotion.

Regarding performance, we utilized the Java Microbenchmark Harness¹⁹ (JMH) and the Berlin SPARQL Benchmark (BSBM) [14] to examine I/O speed of our system compared to TDB2. BSBM provides pre-generated datasets [15] with different scale factors from which we use 200 (75 550 triples) and 2000 (725 305 triples). Even at scaling factor 2000, loading performance is roughly the same with 3.698 ± 0.378 seconds, leading to a writing speed of 196 134 triples per second. Moreover, BSBM’s test driver is used to compare reading performance using 12 SPARQL queries in a mixture. The average time to run the queries is 0.540 ± 0.125 seconds for our system and 0.051 ± 0.008 seconds for TDB2. Results show that our prototype is around 10 times slower than TDB2. An inspection using a Java profiler indicated that further optimization of the internal SQL SELECT queries and database indices is necessary. The benchmarks were executed with one warmup iteration and five measurement iterations on a Lenovo ThinkPad P15v Gen 3 workstation powered by a 12th-generation Intel Core i7-1270P CPU.

All JUnit test cases and benchmark results are available in the GitHub repository.

5. Conclusion and Outlook

This paper presented a first version of a quadstore tailored for Corporate Memories. Our initial implementation comprises RDF management with an SQLite backend, Discretionary Access Control on graph and resource level, confidence-attributed triples, RDF patches and virtual graphs. This is accompanied with a server component providing endpoints and corresponding frontends. Preliminary evaluation verified the correct operation of the system and indicated moderate performance.

In the future, we plan to continue extending and enhancing its features to move closer to our envisioned ATIC system. To improve query performance, a multi-model indexing component is in

¹⁶<https://yaml.org/>

¹⁷A class was selected whenever it tests general behavior of an RDF dataset or graph.

¹⁸<https://github.com/apache/jena/tree/99267df18097141bd2ac27e2b14edc9/jena-tdb2/src/test/java/org/apache/jena/tdb2>

¹⁹<https://github.com/openjdk/jmh>

development. Following recent advances in neuro-symbolic AI, we aim to deploy agents that leverage knowledge graphs within Corporate Memory settings building upon the ATIC system.

Acknowledgments

This work was funded by the project IDIAL (Grant no. 86002329), which is co-funded by the European Union and the state of Rhineland-Palatinate.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT to: Grammar and spelling check, paraphrase, and reword to improve the writing style. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

- [1] World Wide Web Consortium, RDF 1.1 Primer, 2014. URL: <https://www.w3.org/TR/rdf11-primer/>.
- [2] World Wide Web Consortium, SPARQL 1.1 Query Language, 2013. URL: <https://www.w3.org/TR/sparql11-query/>.
- [3] D. Arndt, J. Broekstra, B. DuCharme, O. Lassila, P. F. Patel-Schneider, E. Prud'hommeaux, T. Thibodeau, B. Thompson, Rdf-star and sparql-star, W3C RDF-DEV Community Group Final Report, 2021. URL: <https://w3c.github.io/rdf-star/cg-spec/2021-12-17.html>, final Community Group Report, 17 December 2021.
- [4] C. Jilek, M. Schröder, H. Maus, S. Schwarz, A. Dengel, Towards self-organizing personal knowledge assistants in evolving corporate memories, CoRR abs/2308.01732 (2023). [arXiv: 2308.01732](https://arxiv.org/abs/2308.01732).
- [5] C. Jilek, Self-organizing Context Spaces to Support Information Management and Knowledge Work, Ph.D. thesis, Kaiserslautern University of Technology, Germany, 2024. URL: <https://kluedo.ub.rptu.de/frontdoor/index/index/docId/7939>.
- [6] W. Ali, M. Saleem, B. Yao, A. Hogan, A. N. Ngomo, A survey of RDF stores & SPARQL engines for querying knowledge graphs, VLDB J. 31 (2022) 1–26. doi:10.1007/S00778-021-00711-3.
- [7] O. Erling, I. Mikhailov, Virtuoso: RDF support in a native RDBMS, in: R. D. Virgilio, F. Giunchiglia, L. Tanca (Eds.), Semantic Web Information Management - A Model-Based Perspective, Springer, 2009, pp. 501–519. doi:10.1007/978-3-642-04329-1_21.
- [8] S. Kirrane, A. Mileo, S. Decker, Access control and the resource description framework: A survey, Semantic Web 8 (2017) 311–352. doi:10.3233/SW-160236.
- [9] S. Kirrane, A. Abdelrahman, A. Mileo, S. Decker, Secure manipulation of linked data, in: The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I, Lecture Notes in Computer Science, Springer, 2013, pp. 248–263. doi:10.1007/978-3-642-41335-3_16.
- [10] Y. Chen, X. Duan, Y. Guo, Glr: Graph chain-of-thought with lora fine-tuning and confidence ranking for knowledge graph completion, Applied Sciences 15 (2025). URL: <https://www.mdpi.com/2076-3417/15/13/7282>. doi:10.3390/app15137282.
- [11] G. Xiao, L. Ding, B. Cogrel, D. Calvanese, Virtual knowledge graphs: An overview of systems and use cases, Data Intell. 1 (2019) 201–223. doi:10.1162/DINT_A_00011.
- [12] L. M. Dusseault, HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV), RFC 4918, 2007. doi:10.17487/RFC4918.
- [13] World Wide Web Consortium, JSON-LD 1.1 - A JSON-based Serialization for Linked Data, 2020. URL: <https://www.w3.org/TR/json-ld11/>.
- [14] C. Bizer, A. Schultz, The berlin SPARQL benchmark, in: A. P. Sheth (Ed.), Semantic Services, Interoperability and Web Applications - Emerging Concepts, CRC Press, 2011, pp. 81–103. doi:10.4018/978-1-60960-593-3.CH004.

- [15] T. Blume, Berlin sparql benchmark (bsbm): Evolving graph simulation, 2020. doi:10.5281/zenodo.5714035.