Idiomatic Persistence and Querying for the W3C Web Annotation Data Model

Emanuel Berndl, Kai Schlegel, Andreas Eisenkolb, and Harald Kosch

University of Passau, Chair for Distributed Multimedia Systems,
Innstraße 43, 94032 Passau, Germany
{emanuel.berndl,kai.schlegel,andreas.eisenkolb,
harald.kosch}@uni-passau.de

Abstract. W3C Web annotations are a powerful way to support metadata information about digital resources. The Web Annotation Data Model proposes standardised RDF structures that express this by implementing a hierarchical annotation structure. Those annotations are designed to be shared, linked, tracked back as well as searched and discovered across different peers. However, non-Semantic Web experts may struggle to produce the corresponding RDF data or SPARQL queries. Therefore, we propose Anno4j, a Java-based library that gives developers the possibility to create and consume Web Annotations by using plain old Java objects. Anno4j follows natural Object-oriented idioms including inheritance, polymorphism, and composition to facilitate the development with Web Annotations. An extensible and modular architecture supports enhancements and use-case specific model alterations, while the plugin functionality of Anno4j allows to enrich querying by adding custom function evaluators.

Keywords: Semantic Web, Linked Data, Web Annotations, Java, Developer Tool

1 Introduction

The Web Annotation Data Group[1] recently issued a new version of the Web Annotation Data Model [6]. The central concept of their model is the **Annotation**, being used as a way to give further details, description, or information about another "thing", in general digital resources. Examples could be a comment or tag on a web page or image, or a blog post about a news article. One annotation is devised out of three core components: the **body**, which contains the actual content of the annotation, the **target** specifying the "thing" that the annotation is about, and a **annotation node** itself, which joins the body and the target, while supporting provenance information of the whole annotation. This splits the content from its context in a modular way, which fosters the fundamentals of the annotation model: the designed annotations are interoperable and can be used and interpreted at various different locations, offering the possibility to

[1] https://www.w3.org/annotation/

combine information and metadata beyond the boundaries of single enterprises or applications. The resulting combined knowledge base increases the benefit as well as the degree of information of every participant.

This aligns heavily with the "purpose" of the Semantic Web, with its promising advantages of combined and linked data. Although the advantages are promising, there exists an initial hurdle to make oneself familiar with the Semantic Web technologies as well as Linked Data "rules" [1]. Often times, developers are very skilled in their own respective programming domain, but lack the knowledge of producing RDF and consuming linked data over SPARQL.

To overcome some of these shortcomings and lower the barrier of Semantic Web technologies, Anno4j[2] provides a Java library to directly map Java objects to and from the Web Annotation Data Model. The core contributions are as follows:

– Idiomatic access to W3C Web Annotation Data Model following natural object-oriented idioms.
– Support for use-case specific Web Annotation Model alterations and extensions.
– A library built on-top of OpenRDF Alibaba[3], allowing for a broad field of application.
– Path-based query criteria for extensive annotation search functionality.
– Developer-friendly Open-Source Apache V2 license.

The remaining paper is structured as follows. Section 2 briefly lists the core features of the library and section 3 highlights related work in the context of the WADM. Section 4, section 5, and 6 will give more detailed information about persistence, querying, and deeper insight into Anno4j. Finally, section 7 will conclude the paper by depicting future work and a planned roadmap.

2 Overview

Anno4j is a Java-based library, that offers developers the opportunity to easily create and consume annotations conform to the WADM by writing Java POJOs. The framework has been designed in a modular and extensible fashion, allowing users to extend at every feature of Anno4j. The core functionalities of Anno4j are:

– **Persistence**: Simple Java objects provide the basis of persistence and can easily be created and persisted with a given Anno4j object (see section 4).
– **WADM implementations**: Built-in and predefined implementations for all basic classes proposed by the W3C Web Annotation Data Model.
– **Querying**: A `QueryService` object created by an Anno4j instance can be supported with different query criteria (formalised as LDPath arguments) to query and consume respective annotations of the Anno4j database (see section 5).

[2] `https://github.com/anno4j/anno4j`
[3] `https://bitbucket.org/openrdf/alibaba`

- **Plugin Extensions**: By supporting a plugin interface, users can define own RDF statements in combination with respective evaluation operators, which then can be used as querying criteria in order to even enhance the querying functionality and fine tune it to their particular use-case (see section 6).
- **Context awareness**: RDF databases are often using different contexts to divide their data into subgraphs. This feature is also possible in Anno4j, turning RDF triples into quads.
- **Input and Output**: Anno4j is able to both read and write annotations from and to different standardised serialisations, such as JSON-LD, TURTLE, N3-Triples, RDF/XML, etc.

3 Related Work

Alongside the WADM, the Web Annotation Working Group also issued a protocol, namely the Web Annotation Protocol WAP [4]. The purpose is to provide a standard set of actions which are to be supported by both an annotation client and annotation server in order to cooperate smoothly. This enables the formerly discussed advantages of the WADM to be fully exploited, as it is not just desired to build up pairs of participants, but rather a client and server architecture that allows multiple users to consume the information of single data sources. The protocol makes use of the Linked Data Platform LDP [7] to define their core concept of an **annotation container**. Supported REST and HTTP functionality offers different methods to create and easily query annotations from a given container, and provide information about the container itself. This allows to further familiarise a client with the information or its structure that a given server will support. Listing 1.1 shows an exemplary POST request to create an annotation. The desired annotation content is supported as JSON, in this case a simple annotation that has a `oa:EmbeddedContent` body node containing the String (relationship `rdf:value`) “I like this page!”. The target of the annotation is the homepage “http://www.example.com/index.html”.

Listing 1.1. Example POST request to create an annotation using the WAP protocol

```
POST /annotations/ HTTP/1.1
Host: example.org
Content-Type: application/ld+json

{
  "@context": "http://www.w3.org/ns/oa",
  "@type": "oa:Annotation",
  "body": {
    "@type": "oa:EmbeddedContent",
    "value": "I like this page!"
  },
  "target": "http://www.example.com/index.html"
}
```

Before the WAP has been fully published, similar approaches have been made in order to give the annotations of the WADM (or in this case formerly known as the Open Annotation Data Model OA [5]) a new facet of interoperability by making them shareable more easily over the web. In [3] Pyysalo et al. describe a minimal web interface using REST, that allows users to create and share OA annotations. They implement two core components, the “OA Store” as client and the “OA Explorer” as server respectively, that can further be enhanced with two components for validation and format conversion.

In comparison, the REST-based approaches offer some advantages. As it is a protocol, there exists the freedom of implementing only parts of the specification, allowing the WAP-conform server or client to be adapted to a specific use case and thusly be more lightweight. The REST interface enables the server to be used platform independently, annotations are queried as well as created using JSON. On the other hand, all RDF instances in Anno4j are present as a POJO, allowing them to be used further in object-oriented ways, without the need of up-front parsing. Anno4j is also able to read and generate different serialisations of the RDF objects if needed.

4 Persistence

Anno4j’s persistence implementation follows a very simple mechanism. The library allows direct mapping plain old Java objects to RDF information from a remote or local SPARQL endpoint. Listing 1.2 shows a simple example how to create RDF information with Java objects. The annotation object is created using an Anno4j instance. The resulting RDF triples will automatically include the necessary information, e.g. that the annotation node is of the type (RDF relationship `rdf:type`) `oa:Annotation`. Creating one exemplary body object (type `dctypes:StillImage`) and initialising its field accordingly would result in the RDF entities and relationships shown in figure 1. The RDF triples for properties, types, and relationships between objects are created accordingly after the `persist` was called.

Listing 1.2. Exemplary creation and initialisation of an object

```
// Anno4j object initialised
Anno4j anno4j = new Anno4j();

// Create Annotation object
Annotation annotation =
    anno4j.createObject(Annotation.class);

// Create the body node via Anno4j
ImageDetectionBody body =
    anno4j.createObject(ImageDetectionBody.class);

// Initialise fields
```

```java
body.setDepicts("Barack Obama");
annotation.setBody(body);

// Persist the annotation
anno4j.persist(annotation);
```

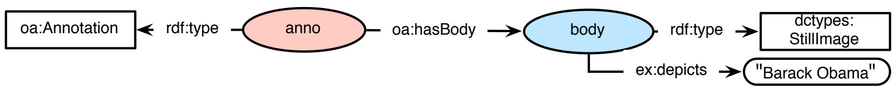

Fig. 1. Resulting RDF graph created with the code shown in listing 1.2. Round and coloured nodes represent instances, a rectangle symbolises an RDF class, and rounded rectangles are RDF literals.

Anno4j already comes with built-in and predefined implementations for most of the classes proposed by the Web Annotation Data Model. This enables developers to start right away with the core functionality of producing annotations. However, use-case specific alterations and enhancements are easy to integrate by extending the persistence layer with their own respective classes. The main element to do this is using the Java annotation `@Iri`, which is used at interface level of the given Java interface and on each setter/getter pair. The annotation at interface level sets the RDF class (relationship `rdf:type`) of the respective RDF object, while an assigned `@Iri` at setter **and** getter method will create the respective predicate attached to the RDF object. Anno4j will automatically provide a proxy-based implementation for the given interface, which already includes the expected behaviour of RDF mapping.

As an example, consider listing 1.3, showing the implementation of a body that is used to convey detected things on a given image and which was already used in the previous example in listing 1.2.

Listing 1.3. Exemplary body implementation with one field "depiction"

```java
// Sets the type of the class
@Iri(DCTYPES.STILL_IMAGE)
public interface ImageDetectionBody extends Body {

    // Setters and getters required for the RDF
       predicate
    @Iri(EX.DEPICTS)
    void setDepicts(String depiction);

    @Iri(EX.DEPICTS)
    String getDepicts();
}
```

5 Querying

Besides persisting Web Annotations, Anno4j also provides ways to query the annotations or even subparts of them that fit specific needs. On the one hand, Anno4j provides convenient mechanisms to directly query e.g. for all annotation bodies with a particular type or Anno4j Java class. On the other hand, Anno4j offers more expressive ways, using the path-based query syntax LDPath[4], to define query criteria to reduce the effort for non-SPARQL experts. LDPath, which is similar to XPath, allows a more compact and inline definition of criteria in contrast to the verbose pattern-based query language SPARQL. A fluent interface API supports readability and comprehensible query definition. A collection of individual criteria defines the desired characteristic of the resulting annotations. Hereby, multiple criteria are combined with a logical `AND` operation.

Listing 1.4 shows an example using LDPath to define two different criteria for annotations. In this example we are searching for annotations which satisfy the conditions that the annotation body should be a `dctypes:StillImage` and Barack Obama is depicted on the image.

Listing 1.4. Anno4j Query Example

```
List<Annotation> annotations = queryService
    .addCriteria("oa:hasBody[is-a dctypes:StillImage]")
    .addCriteria("oa:hasBody/ex:depicts", "Barack Obama")
    .execute(Annotation.class);
```

Although Anno4j uses LDPath as syntax for query criteria, there is no need for a special LDPath-capable RDF endpoint, because LDPath criteria are translated to an equivalent valid SPARQL 1.1 query. This allows developers to reuse generic SPARQL 1.1 endpoints for their use-cases. Besides basic path criteria, Anno4j also supports a wide range of different LDPath condition types[5]:

- Forward and reverse path conditions
- Resolving of namespace abbreviations
- Recursive path like OneOrMore(+) or ZeroOrMore(*)
- Comparison methods like equal, greater, or lower for conditions
- Union of multiple paths
- Type or datatype conditions
- Logical combination of conditions
- Custom functions (see section 6)

After execution of the translated SPARQL query against the specified endpoint, all query results are automatically transformed to corresponding annotated Java objects. This abstraction layer allows developers to easily work with RDF information in contrast to constructing complex SPARQL queries and parse the SPARQL results.

[4] `http://marmotta.apache.org/ldpath/`

[5] For further and detailed description of the LDPath criteria, please refer to the LDPath specification at `http://marmotta.apache.org/ldpath/language.html`

6 Internals and Extensions

At its core, Anno4j builds upon the OpenRDF Alibaba library (former Elmo codebase) which provides simplified RDF store abstractions and combines the flexibility and adaptivity of RDF with the powerful object-oriented programming model of Java. It is able to map Java objects to and from RDF resources in a non-intrusive manner that enables developers to work with resources stored in a SPARQL endpoint. Nested properties of RDF resources are lazy evaluated to avoid unnecessary fetching of unused information for faster and efficient query evaluation. Considering listing 1.3, lazy evaluation implies that the RDF value for `depicts` is not fetched from the SPARQL endpoint until `getDepicts` of the respective Java object is called.

As mentioned before, Anno4j uses LDPath syntax to define query criteria. Internally, LDPath criteria is automatically transformed to valid SPARQL 1.1 syntax. Some LDPath expressions can be directly mapped to similar SPARQL 1.1 property path expressions (e.g. path selection, inverse path selection "`^`", or alternative path "|"). LDPath expressions which can't be directly mapped to SPARQL 1.1. are translated to similar SPARQL constructs (e.g. datatype or "is-a" tests are mapped to SPARQL `FILTER` constructs). To support extensibility, the translation process follows the Interpreter software pattern. Hence there exists a specific interpreter for each LDPath expression. This allows developers to easily integrate custom LDPath expressions, such as function predicates, test functions and filters, as well as register a query interpreter which transforms the new query element to valid SPARQL 1.1 to ensure full compatibility with generic SPARQL endpoints.

This extension mechanism was induced by the requirement to integrate query extensions like the recently proposed SPARQL-MM [2]. SPARQL-MM can, for example, query for temporal or spatial relationships of different annotations. Using the SPARQL-MM extension, Anno4j can be used for semantic querying e.g. to query for all annotations depicting "Barack Obama" positioned left besides "Angela Merkel" on an image. In this case, a custom function `fn:leftBesides` can be used, which does not represent an actual serialised relationship between two annotations, but rather uses annotation information like the target and its selector to evaluate the corresponding function.

7 Conclusion

This paper introduced the library Anno4j, which enables Java developers to create and consume RDF annotations. Those annotations are conform to the WADM, allowing them to be shared and exchanged between different locations. Anno4j features simple persistence of RDF objects via Java objects, its querying functionality is based on LDPath, supporting a wide range of combinable path criteria to form a powerful annotation consumption tool. Both features can be extended easily, so developers can fine tune their respective Anno4j instance to their needs.

Anno4j is available under Apache V2 license at github. Future work on this library will include the attempt to provide the functionality of Anno4j to other programming languages. First proof-of-concepts show positive results for a C++ mapping using Java Native Interface. This would allow developers to write their software natively in C++ but still use Anno4j for a convenient creation and querying of Web Annotations. Other extensions like SPARQL-MM querying or a QueryWizard for a guided creation of LDPath criteria based on statistics of your SPARQL endpoint are currently actively developed by the community.

The current application of the Anno4j library has led to various lessons learned, as different projects make use of Web Annotations in conjunction with Anno4j to make a more convenient use of the annotations. However, as nearly every application of today is web-oriented, a web facet could lift Anno4j to a broader use case. This requirement is exactly tailored to the WAP specification, so future steps will include a layer on top of Anno4j, allowing it to be used as a WAP-conform server. Additionally, as the querying mechanism of Anno4j in combination with LDPath is manifold, we intend to extend the WAP requirements for our implementation to deliver more comprehensive querying.

Acknowledgements

The presented work was developed within the MICO project partially funded by the EU Seventh Framework Programme, grant agreement number 610480.

References

1. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts pp. 205–227 (2009)
2. Kurz, T., Schlegel, K., Kosch, H.: Enabling access to linked media with sparql-mm. In: Proceedings of the 24nd international conference on World Wide Web (WWW2015) companion (LIME15) (2015)
3. Pyysalo, S., Campos, J., Cejuela, J.M., Ginter, F., Hakala, K., Li, C., Stenetorp, P., Jensen, L.J.: Sharing annotations better: Restful open annotation. ACL-IJCNLP 2015 p. 91 (2015)
4. Sanderson, R.: WAP web annotation protocol. W3c working draft, W3C (2015), https://www.w3.org/TR/annotation-protocol/
5. Sanderson, R., Ciccarese, P., de Sompel, H.V.: OADM open annotation data model. W3c community draft, W3C (Feb 2013), http://www.openannotation.org/spec/core/
6. Sanderson, R., Ciccarese, P., Young, B.: WADM web annotation data model. W3c working draft, W3C (Oct 2015), https://www.w3.org/TR/annotation-model/
7. Speicher, S., Arwe, J., Malhotra, A.: ldp linked data platform 1.0. W3c recommendation, W3C (Feb 2015), https://www.w3.org/TR/ldp/